Sub-LANs: An Inexpensive Networking Alternative

June 1988 • $3.95 USA (Canada $4.95)

For the PC Systems Integrator

Micro Systems JOURNAL

Piloting Your DOS Application to the 386 Platform

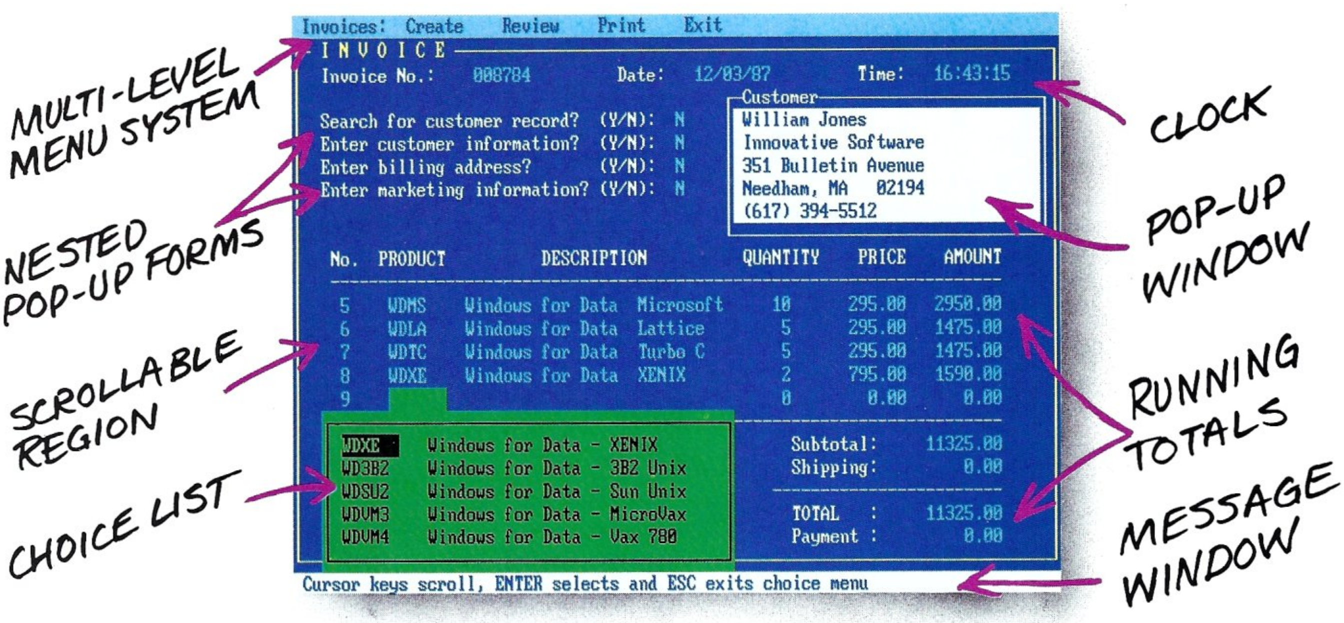
386 Development: Assemblers and Fortrans

Using Direct Memory Access

New Prices

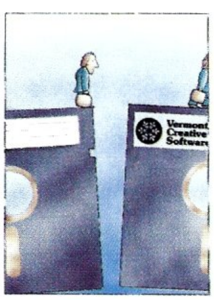
OS/2

WINDOWS FOR DATA®



If you program in C, take a few moments to learn how Windows for Data can help you build a state-of-the-art user interface.

- ☑ **C**reate and manage menus, data-entry forms, context-sensitive help, and text displays — all within windows.
- ☑ **D**evelop window-based OS/2 programs right now, without the headaches of learning OS/2 screen management. Run the same source code in PCDOS and OS/2 protected mode.
- ☑ **B**uild a better front end for any DBMS that has a C-language interface (most popular ones do).



FROM END TO BEGINNING

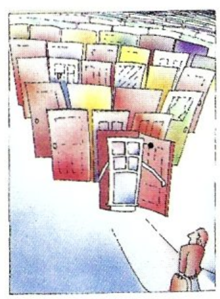
Windows for Data begins where other screen packages end, with special features like nested pop-up forms and menus, field entry from lists of choices, scrollable regions for the entry of variable numbers of line items, and an exclusive built-in debugging system.

NO WALLS

If you've been frustrated by the limitations of other screen utilities, don't be discouraged. You won't run into walls with Windows for Data. Our customers repeatedly tell us how they've used our system in ways we never imagined — but which we anticipated by designing Windows for Data for unprecedented adapatability. You will be amazed at what you can do with Windows for Data.

YOU ARE ALWAYS IN CHARGE

Control functions that you write and attach to fields and/or keys can read, compare, validate, and change the data values in all fields of the form. Upon entry or exit from any field, control functions can call up subsidiary forms and menus, change the active field, exit or abort the form, perform almost any task you can imagine.



OUR WINDOWS WILL OPEN DOORS

Our windows will open doors to new markets for your software. High-performance, source-code-compatible versions of Windows for Data are now available for **PCDOS, OS/2, XENIX, UNIX,** and **VMS.** PCDOS versions are fully compatible with Microsoft Windows. **No royalties.**

MONEY BACK GUARANTEE

You owe it to yourself and your programs to try Windows for Data. If not satisfied, you can return it for a full refund.

Prices: PCDOS $295, Source $295. OS/2 $495. XENIX $795. UNIX, VMS, please call.

Call: (802) 848-7731 ext. 33

Telex: 510-601-4160 VCSOFT FAX 802-848-3502

Vermont Creative Software

21 Elm Ave.
Richford,
VT 05476

Simply the BEST C and Pascal on AT, 386, Sun, Apollo, RT, VAX, 370

"The most rock-solid C compiler in the industry. Superb technical support and portability. Superior code generated."
Gordon Eubanks, Symantec — Q&A (386).

"It simply works, with no trouble, no chasing strange bugs, and excellent warning and error messages ... a professional product."
Robert Lerche, Bay Partners.

"For large-scale software development, the highest quality C compiler available on the market today. Pragmas are great. Quality of support is exceptional." **Randy Neilsen, Ansa—Paradox (DOS,OS/2).**

"15% smaller and 15% faster than Lattice C."
Robert Wenig, Autodesk.

"Our software is running anywhere from 30 to 50% faster than when compiled under Lattice." **David Marcus, Micronetics.**

"We switched from Lattice due to a 10% reduction in code size. The compiler is very stable." **Lee Lorenzen, Ventura Software — Ventura Publisher, marketted by Xerox Corp.**

"Best quality emitted code by any compiler I've encountered. Often amazing." **Bill Ferguson, Fox Software — FoxBase (386).**

"Messages sometimes pointed out type mismatches, incorrect-length argument lists, and uninitialized variables that had been undetected for years [in 4.x bsd]." **Larry Breed, IBM ACIS [RT PC].**

"Diagnostics turned up bugs missed by other compilers. Rapid bug fixes by technical support, someone who knew what he was talking about. 80386 code is well optimized."
Tim Addison, Logistics Data Systems.

"386 protected mode support is fantastic, especially the access to large amounts of memory. It's mainframe compute power on a PC." **Dan Eggleston, Viewlogic.**

"The preprocessor supplied with Professional Pascal is quite useful. The code quality and control over segmentation and memory models are superior to MS Pascal." **Bob Wallace, QuickSoft.**

Check Out These Reviews

- **High C ™:**

Computer Language	February 1986, '87	
Dr. Dobb's Journal	August 1986	
PC Magazine	Jan. 27, 1987	(80386 version)
Dr. Dobb's Journal	July 1987	(80386 version)
BYTE Magazine	November 1987	(80386 version)

- **Professional Pascal ™:**

PC Magazine	Dec. 29, 1985	
Computer Language	May 1986	
PC Tech Journal	July 1986	
Journal of Pascal, Ada, & Modula-2	Nov.-Dec. 1986	
BYTE Magazine	Dec. '86, June '87	(80386 version)

Why MetaWare compilers

- They are specifically designed for serious software developers.
- They are reliable and robust: they don't break at every turn.
- Their generated code is the best, or near best, on each architecture.
- Their superior diagnostic messages help you produce better products more quickly.
- Your source can be ported with ease to the most popular systems.
- You can link mixed-language modules from our compilers, others
- You can benefit from high-level, personal technical support.
- You can take advantage of the latest ANSI C extensions, and/or extensive Pascal extensions. **High C** has been tracking the ANSI Standard for two years; **Professional Pascal** will soon have a VS Pascal compatibility switch and several Apollo Pascal ext'ns.
- You can take advantage of the latest 387 and Weitek 1167 support — we have the only compilers with Weitek real mode support.

Power Tools for Power Users

Ashton-Tate: dBase III Plus, MultiMate; Autodesk: AUTOCAD, AUTOSKETCH (8087, '387, Weitek); Boeing Computer Services (Sun); CASE Technology (Sun); CAD/CAM giant Daisy ('86, '386, VAX); Deloitte Haskins & Sells; Digital Research: FlexOS; GE; IBM: 4.3/RT, 4680 OS; Lifetree Software (Pascal): Volkswriter Deluxe, GEM-Write; Lugaru: Epsilon; NYU: Ada-Ed cmplr; Semantec: Q&A; Sky Computers; ... (Product names are trademarks of the companies indicated.)

Industrial-Strength

MetaWare C and Pascal compilers are designed for professional software developers. These tools are loaded with options to control them for special purposes. You can adjust the space-time trade-off in code quality. You can adjust external naming conventions to agree with linkers and operating systems. You can specify segment names for segmented architectures, and to help place code or data in particular places for embedded applications. You can select from five memory models for the 8086 family. And on and on.

A Partial List of Optimizations

Common subexpression and dead-code elimination, retention and reuse of register contents, jump-instruction size minimization, tail merging (cross jumping), constant folding, short-circuit evaluation of Boolean expressions, strength reductions, fast procedure calls, automatic mapping of variables to registers (where advantageous), ...

"Platform" — Code Quality

Sun,Apollo,SGI — 18%, 3%, 26% > resident compiler (Dhrystone).
PC: DOS, OS/2 — 3-10% > Microsoft C; 30% > MS Pascal, LatticeC.
386 32-bit DOS — no competitors, since November, 1986.
286, 386 UNIX — 66% better than pcc (Dhrystone, 386).
VAX VMS — ≈ DEC's excellent C and Pascal; better features.
VAX Ultrix — 19% > pcc (Dhrystone); much > Berkeley Pascal.
RT PC/4.3bsd — 89% > the original port of pcc (Dhrystone).
370 CMS,UNIX — much better than any C, and VS Pascal.
AMD 29000 — >40,000 Dhrystones! Available in Q2, cross.

(408) 429-6382, telex 493-0879. ***Since 1979.***

Meta Ware ™
INCORPORATED
903 Pacific Avenue, Suite 201 • Santa Cruz, CA 95060-4429

The Clear Choice for Large Programming Projects — PC Tech J.

We've spent years developing our file manager so you won't have to.

You could invest hundreds of programming hours writing a file management system for your next application.

Or you could simply invest in Btrieve® And get all the file handling functionality you need, through subroutine calls from your favorite programming language.

Portable. Write your application once. Wherever Btrieve runs, your application will run, whether in a single user or multiuser environment. In fact, Btrieve is the standard access to NetWare®

Fast. Written in assembly language, Btrieve uses b-tree indexing algorithms with caching and automatic balancing for fast, efficient file management.

Safe. Btrieve is the only fault tolerant file manager with built-in file recovery. In the event of a system or power failure, your database is protected.

Flexible. Develop applications with the capabilities you need most. Like 255 open files, unlimited records per file, 24 indexes per file, and a maximum file size of up to four gigabytes. You can access Btrieve from BASIC, C, Pascal, COBOL and others.

Invest in Btrieve. At just $245 for single user and $595 for multiuser, it's a small price to pay for all the file manager you'll ever need.* And you'll never pay royalties on the applications you develop. To find out more, see your authorized Novell reseller, or call (512) 346-8380.

For more information, call from your modem 1-800-444-4472 (8 bit, no parity, 1 stop bit) and enter the access code NVBT13.

NOVELL

For software solutions, you should be seeing red.

*Suggested retail price (US dollars) World Headquarters, 122 East 1700 South, Provo, Utah 84601 (801) 379-5900
Requires PC-DOS or MS-DOS 2.X, 3.X or Xenix.

CIRCLE 65 ON READER SERVICE CARD

For the PC Systems Integrator

Micro/Systems JOURNAL

June 1988
Vol. 4 . No. 6

FEATURE ARTICLES

PRODUCT REVIEWS

COLUMNS

About the cover: You're a Top Gun of computer integration, right? Your first mission: To adapt current DOS applications to use the new 80386 platform. Your ongoing assignment: To tap the power offered by the 386 chip to deliver faster, more powerful software solutions to integration problems. In this issue we will help you meet both of these challenges. Our lead story discusses how to port programs to the 386 protected mode. We also offer a look at new 386 developer's tools and control programs.
Cover photograph by Michael Carr
Model supplied by Richard Hermle.

DEPARTMENTS

Micro/Systems Journal (ISSN 8750-9482) is published monthly by M & T Publishing, Inc., 501 Galveston Drive, Redwood City, CA 94063; (415) 366-3600. Second-class postage paid at Redwood City and at additional entry points. **POSTMASTER:** send address changes (form 3579) to *Micro/Systems Journal*, Box 3713, Escondido, CA 92025. **Change of Address:** Please send old label and new address to *Micro/Systems Journal*, Box 3713, Escondido, CA 92025. **Customer Service:** For subscription problems, call: (800) 321-3333; In California call (619) 485-9623. **Subscription Rates:** U.S., $29.95 for one year, $56.97 for two years. Canada and Mexico add $17 per year for airmail, $7 per year for surface mail. Other countries add $28 per year for airmail.

Introducing DESQview™ 2.0 API Tools

DESQview API Reference Manual

This is the primary source of information about the DESQview API. It contains all you need to know to write Assembly Language programs that take full advantage of DESQview's capabilities. The Reference manual comes with an 'include' file containing symbols and macros to aid you in development. AVAILABLE NOW!

DESQview API C Library

The DESQview API C Library provides C Language interfaces for the entire set of API functions. It supports the Lattice C, Metaware C, Microsoft C, and Turbo C compilers for all memory models. Included with the C Library package is a copy of the API Reference Manual and source code for the library. AVAILABLE NOW!

DESQview API Debugger

The DESQview API Debugger is an interactive tool that enables the API programmer to trace and single step through API calls from several concurrently running DESQview-specific programs. Trace information is reported symbolically along with the program counter, registers, and stack at the time of the call. Trace conditions can be specified so that only those calls of interest are reported. AVAILABLE JUNE 88!

DESQview API Panel Designer

The DESQview API Panel Designer is an interactive tool to aid you in designing windows, menus, help screens, error messages, and forms. It includes an editor that lets you construct an image of your panel using simple commands to enter, edit, copy, and move text, as well as draw lines and

Bringing new power to DOS

boxes. You can then define the characteristics of the window that will contain the panel, such as its position, size, and title. Finally, you can specify the locations and types of fields in the panel.
The Panel Designer automatically generates all the DESQview API data streams necessary to display and take input from your panel. These data streams may be grouped together into panel libraries and stored on disk or as part of your program. AVAILABLE JUNE 88!

DESQview API Pulldown Menu Manager

The DESQview API Pulldown Menu Manager is an interactive tool to aid you in designing pulldown menus. This DESQview API tool assists you in giving your DOS program an OS/2-like look and feel. AVAILABLE JULY 88!

DESQ™ view API Conference
August 16, 17, 18
Marina Del Rey, California

Call for registration information (213) 392-9851

Quarterdeck

Quarterdeck Office Systems
150 Pico Boulevard
Santa Monica CA 90405
(213) 392-9851

From the Editor's Desk

IBM Fights Back

A year has passed since IBM introduced its new PS/2 line of computers, discontinuing the PC line that it had marketed for almost six years (a record for IBM). Its April 1987 introduction comprised of three machines, the Models 30, 50, and 60. The Model 30 is essentially a repackaged version of the old XT; the Model 50 is an upgraded replacement for the AT; and the Model 60 is an enhanced version of the 50. Later in the year, IBM introduced the Models 25 and 80—the Model 25 being a stripped down version of the 30 while the Model 80 is a powerful 386-based machine designed to compete with Compaq's highly successful 386 machine (which had been on the market for a full year at the time of the introduction).

IBM has been successful in selling a large number of Model 30 and 50 machines, primarily to its traditional customers—Fortune 1000 companies. The machines are being sold on price, not features. IBM has found that many of its traditional customers are continuing to purchase the much lower cost, older PC-compatibles, or turning to companies such as Compaq and AST that have gained a reputation for providing products that deliver performance equal to or better than the PS/2 machines using the older basic technology.

IBM has watched its share of the desktop market shrink from over 56 percent to an estimated 47 percent over the last year, as Compaq's share has doubled to 10 percent and the combined share of the other compatibles has increased from 19 to 24 percent. If this trend were to continue, IBM could soon be ranked as a minor player in the desktop computer marketplace.

IBM is now faced with the prospect that its competitors are implementing OS/2 on their PC-compatible 286- and 386-based machines. And even more frightening is the appearance of PS/2 compatibles.

IBM has a reputation as the most aggressive marketer of computer systems in the world. IBM typically responds quickly to competition, mainly keeping competitors off balance by introducing a major new computer line every four years. However, the PC's success must have gone to their heads. They waited six years before making the PC obsolete. But they appear to have learned their lesson.

Less than one year after introducing the PS/2 line, IBM announced that it would soon make major changes in the line. IBM is threatening to replace the Models 25 and 30 with new models containing 80286 processors, more memory, and the MicroChannel Architecture. (This probably means that the price for the Models 50 and 60 will be cut to that currently asked for the Models 25 and 30). This means that the PS/2 entry-level machines (which currently sell for between $1,000 and $3,000) will be capable of running OS/2. It also means that these units will be able to handle MCA multimaster plug-in cards, such as high-performance networking interface cards. The PS/2's MCA multimaster feature could also be utilized to improve the performance of hard disk and display systems.

And IBM has indicated that new PS/2 systems in the current Models 50 and 60 price range will be 80386-based systems. This means that the prices for this class of performance will be slashed by 40–60 percent.

When IBM introduced the PS/2 line, it also introduced a new display system, the VGA. VGA clones are only just now reaching the market and few application software packages take advantage of the VGA's superior performance. Yet IBM has announced that next year it will introduce a new, even higher performance display system. Use of OS/2 with the Presentation Manager video display interface should make porting software to the new system much easier, which should make things even more difficult for IBM's competition.

It appears that IBM is becoming more innovative as well as price competitive. The question is whether we are again being offered promises rather than actual substance. If IBM delivers as it is promising, the competition could be in trouble. If IBM does not deliver as promised, it will no doubt lose even more market share and possibly even lose its position as the market leader.

Sol Libes

For the PC Systems Integrator
Micro/Systems JOURNAL

EDITORIAL
Founder and Editor Sol Libes
Technical Editors Stephen R. Davis, Don Libes
Associate Editors Lennie Libes, Susan Libes
Contributing Editors A.G.W. Cameron, Patrick Corrigan, P.L. Olympia
Managing Editor Thomas M. Woolf

PRODUCTION
Art & Production Director Larry L. Clay
Art Director Kobi Morgan
Asst. Art Director Barbara Mautz
Typographer Lorraine Buckland

CIRCULATION
Director of Circulation Maureen Kaminski
Asst. Circulation Mgr. Andrea Weingart
Newsstand Mgr. Sarah Frisbie
Direct Mktg. Specialist Kathleen Shay
Fulfillment Coord. Francesca Martin

ADMINISTRATION
V.P. Finance & Operations Kate Wheat
Business Mgr. Betty Trickett
Accounting Supv. Mayda Lopez-Quintana
Accts. Payable Asst. Luanne Rocklewitz
Accts. Receivable Asst. Wendy Ho

ADVERTISING
Advertising Director Richard Mixter
National Account Mgr. Dwight Schwab
National Account Mgr. Tami Brenton
Advertising Coord. Shaun Hooper

M&T Publishing, Inc.
Chairman of the Board Otmar Weber
Director C.F. Von Quadt
President & Publisher Laird Foshay
V.P. of Publishing William P. Howard

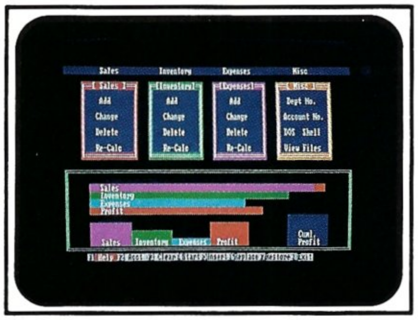
Micro/Systems Journal (ISSN 8750-9482) is published monthly by M & T Publishing, Inc., 501 Galveston Drive, Redwood City, CA 94063; (415) 366-3600. Second-class postage paid at Redwood City and at additional entry points.

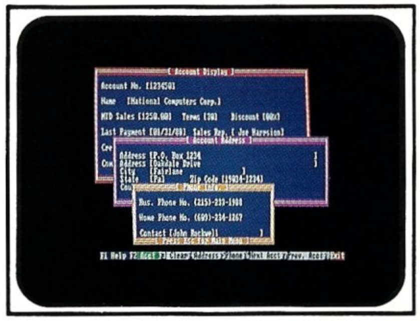
Article Submission: If you have a specific area of expertise or interest and would like to contribute, please write Micro/Systems Journal, P.O. Box 1192, Mountainside, NJ 07092; (201) 522-9347, or contact M & T Publishing, Inc., 501 Galveston Drive, Redwood City, CA 94063; (415) 366-3600. Please do not submit articles without first contacting the editors. Author's guidelines available upon request.

Correspondence: Please send letters to the editor to Micro/Systems Journal, 501 Galveston Drive, Redwood City, CA 94063. Other editorial correspondence may also be directed to P.O. Box 1192, Mountainside, NJ 07092. The editors may also be reached via MCI Mail (SLIBES or MSJ).

Advertising Rates: Available upon request. Call (415) 366-3600 or write to: Advertising Department, Micro/Systems Journal, 501 Galveston Drive, Redwood City, CA 94063.

Don't fall flat on your Ashton.





Power up Q-PRO 4™. Get all the reliable, 'bug-free' features you need NOW!

The Most Powerful Screen Handler You've Ever Seen.

Q-PRO 4 offers all the features, the power, and the reliability you need. And it's available now!

- What you see is what you get (WYSIWYG) no coding
- Fill-in-the-blanks screens
- Field editing with attributes
- Entry fields addressable as variables in the program
- Yes/no fields
- Multiple choice fields
- Entry fields symbolically addressable
- Programmable screen field types
- Picture data editing
- 16 color palette
- Active field color
- Painted background

Fully Featured Windows (optional)

- 100 windows per screen
- Multiple windows displayed simultaneously
- Frame and title
- Any size, any location
- Create WYSIWYG with included editor
- 3 Types: data entry, scrolling help, and data display

No Limits

Unlimited files open simultaneously (we have overcome the DOS limits).

- 2,000,000,000 bytes per file
- Unlimited record size
- Unlimited number of fields on screen
- Unlimited memory arrays
- Unlimited size of dimensioned memory arrays

Extraordinary Data Files

- Data dictionary
- Nine indexes per data file
- All indexes in one file
- Ascending and descending indexes
- B+ tree (CBTREE) state-of-the-art speed
- Also random, ASCII, and comma delimited

Advanced Fourth Generation Language

- Event driven architecture
- Procedural, with over 120 commands
- Lightbar menus
- Global subroutines
- Global variables
- Directory and path management
- Two dimensional tables
- Variables: string, numeric, floating point, Boolean
- DOS shell
- Random numbers
- On line debugger
- Julian dates
- European dates and currency formats
- Full math package
- Sophisticated editor included

Modern Transaction Processing

- Transaction recovery
- Transaction rollback

Security

- Optimizer encrypts and speeds up programs
- Data encryption
- Screen security
- Easy, consistent, predictable, corrupted file recovery

LAN and Multiuser

Record and file lock for:

- Novell Netware
- IBM PC-Network
- 3COM
- Software Link PC-MOS

Join the next generation! Order your Q-PRO 4 today.
Only $795.00 with windows.
$495.00 without.

q-n-e™
international

136 Granite Hill Court
Langhorne, PA 19047

(800) 333-0448

TLX 291765

News & Views

by Sol Libes

Random Rumors & Gossip

There is increasing talk of an 80386 clone chip being introduced by a manufacturer other than Intel. After all, NEC produces the V20 and V30 chips that are enhanced 8088/8086 clones, thus proving that it can be done. The big hurdle is legal rather than technical.

According to data from **International Data Corporation**, a respected market researcher, the Compaq 386 machines are outselling IBM's 386-based systems. And if one takes into account all the other producers of 386 PC-compatibles, it means that IBM is only a minor player in the 386 market. It is becoming increasingly apparent that machines using the old AT architecture are providing performance equal to or better than that of the PS/2 with its much-touted (and patented) MicroChannel Architecture.

While many companies are just getting into production of 1-Mbit dynamic RAM ICs, word has come that several companies will begin sampling 4-Mbit chips next year and 16-Mbit chips in 1990. Initial production is expected to follow about a year later.

Look for **DEC** to follow in Texas Instruments' footsteps with a private-labeled version of the Macintosh. **IBM** already has one private-label arrangement for the PS/2 in England and is expected to enter into more such arrangements on the European continent; but don't look for any such arrangements here in the United States where IBM's PS/2 market share is higher.

The **National Computer Conference** (NCC) is no more. NCC used to be the largest and best-attended annual computer show, catering to MIS/DP mainframe computer users. Personal computer shows like Comdex have put NCC out of business. Is this a message for the future for MIS/DPers?

Release 4.0 of UNIX System V is expected shortly from **AT&T.** It should incorporate features from Microsoft's XENIX, Sun's SunOS, and the Berkeley 4.2 and 4.3 systems. Look for it to also add real-time capabilities, system administration improvements, enhanced networking, and features specifically for the international market.

Intel has stated that it is not yet shipping production quantities of 80386 chips rated at 25 MHz and that only 20-MHz chips are available. However, several companies are shipping 25-MHz 386 systems. All claim they are using 20-MHz chips that they have tested and found to run reliably at 25 MHz.

Motorola will soon begin shipping a 33-MHz version of the 68020, a 16-MHz 68000, and a 16-MHz 68HC000. The new 68020 will no doubt end up in high-end workstations from Apple, Sun Microsystems, and Apollo Computer. The new 68000 and 68HC000 are most probably destined for a new, high-performance Macintosh and a portable Mac.

Apple Computer is rumored to be readying a 1.6-MB floppy drive for a new version of the Macintosh due out shortly. A facility to read IBM 1.44-MB disks also is expected either from Apple or third parties.

Hayes Microcomputer Products is expected to shortly introduce a new 9600-baud modem that conforms to the V.32 standard. This will make V.32 the most popular of the 9600-baud protocols. Several V.32 modem makers have also reduced prices to well under $2,000, with discounted prices expected to be close to the $1,000 mark. These moves are expected to finally generate a true standard in the 9600-baud marketplace.

Macintosh clone prototypes have reportedly been shown in Brazil and the Far East. No word as yet on whether they will go into production, and even if they do whether they would be imported into the U.S. There are also rumors that a Macintosh software emulator that runs on 680x0-based UNIX systems is in development.

OS/2 News

The first product to make OS/2 support multiuser operation has been introduced by **TPS Systems**, San Antonio, Texas. Called TPS/2, it will support up to 48 terminals. No doubt we will see several more multiuser extensions to OS/2 shortly.

IBM has demonstrated the OS/2 Extended Edition Version 1.0 at several conferences and to key corporate customers and is expected to begin shipments next month. This early version is expected to lack the Presentation Manager and LAN Server components, which should become available with Version 1.1 which is due out in the fourth quarter.

Microsoft began shipping the Presentation Manager (PM) System Developer's Kit (SDK) to software developers early in April, about four months behind schedule. The SDK includes a working "early" beta version of the PM (final beta copies are promised for August). Microsoft is cautioning developers that there are still bugs in the PM, which they promise will be worked out before it is formally released. Release of the product has been rescheduled to October, but many feel it will be delayed even further. In the meantime, Microsoft cautions developers not to use certain functions. Microsoft is also known to be working hard at improving the response time of the PM.

The SDK comes in a suitcase-sized carton that includes five boxes of disks, with each box containing typically 10 disks.

PS/2 Clone Update

By the time this is published, **Dell Computer** and **Tandy** are expected to boast that they are the first companies out with MicroChannel Architecture (MCA)-based PS/2 compatibles. What is more notable is that companies such as Compaq, Zenith, and AST Research, who command the lion's share of clone market, have decided to hold off bringing out PS/2 MCA clones. The reason is obvious—there is a noted lack of demand for MicroChannel systems. The cloners have effectively demonstrated that they can equal or exceed MCA performance using the existing, non-proprietary AT bus. Current buyers of MCA-based systems are buying the IBM name rather than the technology, hence MCA clones will be a hard sell. This is especially true when IBM's licensing fees will prevent any significant price difference between a clone and an IBM machine. Thus the PS/2 cloning efforts appear more a desire for prestige rather than a serious effort to sell PS/2 MCA machines. Do not expect to see a significant number of PS/2 clones being shipped this year or next.

80386/486 News

Intel has begun disclosing more information on its new 80486, the successor to the 80386. Engineering samples are expected to be shipped before year-end, with production to begin in 1990. The device will be upward-compatible with the 386, and it will provide increased processing speed, improved direct-memory access functions, and a virtual 80286 mode (the 80386 provides only a virtual 8086).

Intel has finally released the 80387 chip. Long rumored to be an 80386 pin-compatible replacement for the 80286, it has turned out to be aimed at controller applications requiring 32-bit

. . . continued on page 10

Continued from page 6
processing. The chip lacks such 386 features as real mode, paging, and 16-bit code.

MS-DOS Cloning

Phoenix Technologies, the company that made its mark generating an IBM-PC-compatible ROM BIOS, has acquired Paterson Laboratories, formerly a division of Microsoft. Paterson has been shipping an MS-DOS clone to PC clone makers. Phoenix will now be competing with Microsoft in the DOS marketplace. The question now is whether Phoenix will offer DOS enhancements and whether they plan to also offer an OS/2 clone?

Presentation Manager & X-Windows Merger?

X-Windows is the graphical interface developing standard for UNIX systems developed by MIT and DEC, and it is being endorsed by virtually all the UNIX vendors. It should be available on systems from DEC, Sun, AT&T, Apple, and many others. It is essentially a toolkit and windowing manager. However, the visual screen presented to the user is left to the system vendor to create. X-Windows is designed to work in a multiuser distributed processing networking environment.

On the other hand, the IBM/Microsoft Presentation Manager, which is still in development, is a complete, specific, visual user interface similar to Microsoft Windows with a rich set of graphics functions. It also is a fixed environment and hence is easier for applications programmers to deal with. It is tied directly to OS/2, a single-user operating system, and needs other facilities to communicate with programs on other systems.

There are rumors that Microsoft is working with Sun Microsystems to graft the PM on to X-Windows using Sun's NeWS (Network-extensible Windowing System) graphical engine. Microsoft has already bought an unlimited license for the NeWS code from Sun. NeWS offers built-in PostScript display facilities, is compatible with X-Windows, and provides a richer graphics toolkit than X-Windows.

The PM/NeWS marriage would establish the Presentation Manager on systems from Apple, DEC, AT&T, Sun, *et. al.* (probably to the dismay of those system owners), and provide a more powerful platform for the PM than that provided by OS/2.

Compaq Leads in 386 System Sales

It is estimated that by the end of the first quarter of this year, close to one million 386-based systems were sold with **Compaq** having a 70 percent share of the 386 market. **IBM**, on the other hand, introduced its 386-based system, the Model 80, seven months after Compaq's introduction and because of delayed shipments lost about a year in the market. Also, because of the lack of third-party PS/2 boards, IBM is currently a minor player in the 386 marketplace.

Atari Introduces UNIX Workstation

Atari has been demonstrating its new 68030-based UNIX workstation and is promising to begin initial shipments next month. Expected to sell for under $5000, it will feature UNIX System V, Version 3.1, have 4-MB of RAM, plus another megabyte of video RAM, a 44-MB removable-media Winchester drive. It will use the VME bus for expansion, and support communications via the ISO model along with Sun's NFS networking. An Ethernet interface option will be provided.

The motherboard will also contain an Inmos T-800 Transputer. X-Windows will be supported on displays of 1280 × 960, 1024 × 768, and 640 × 480 color graphics. Atari is also promising an upgrade path for present Atari ST system owners to the new system.

The system is expected to be marketed primarily in Europe and to compete primarily with Apple's Mac-II and Sun's systems. □

Break the 640K Barrier for $59.95.

Introducing Quarterdeck Expanded Memory Managers™: QEMM 386 and QEMM 50/60.

Your 80386 PC, IBM® Personal System/2™ Model 80, PC or AT with 80386 add-in board, as well as your IBM Personal System/2 Models 50 or 60 can all break through the DOS 640K barrier. Now you can have maximum use of your memory—whether you have one megabyte or 32— with the Quarterdeck Expanded Memory Manager. All without having to purchase special expanded memory boards.

Unlock hidden potential beyond 640K.

QEMM uses hidden features within your existing memory to make it compatible with the Lotus-Intel-Microsoft Expanded Memory Specification (EMS) version 4.0.

Now you can run colossal spreadsheets, databases, and CAD models designed for expanded memory. Using Lotus 1-2-3, Symphony, Framework, Paradox, AutoCAD, Microsoft Excel and more.

Improve productivity with DOS multitasking.

And if you'd like to use these programs all together —multitasking beyond 640K— QEMM extends the capabilities of our popular DESQview™ multitasking environment.

PC MAGAZINE EDITOR'S CHOICE NOV. 24, 1987

DESQview lets your programs work together in a familiar way, giving you many of the capabilities and features only promised by other systems. In fact, the editors of *PC Magazine* recently named it "Best Alternative to OS/2," and *InfoWorld* gave it a report card rating of 9.1.

We offer productivity solutions for the forgotten 12 million.

If you are one of the 12 million or so 8088, 8086 or 80286 PC users who feel left out of this new world, don't despair. We have options that let you keep your computer and favorite programs and give you today what the newest computers and operating systems are promising for the future.

Visit your dealer for more information on Quarterdeck products. And ask to see the chart we've prepared explaining today's maze of memory options. and how you can break the 640K barrier.

For 386 PC, IBM PS/2 model 80 and 80386 add-in board users, QEMM/386:

- Fully supports the Lotus-Intel-Microsoft Expanded Memory Specification (EMS) version 4.0
- Enables the 80386 user to configure the 80386's memory beyond the first megabyte as expanded memory or as a combination of both expanded and extended memory.
- Fills out missing DOS memory to 640K and beyond by mapping into unused video memory. When used with a CGA graphics adapter this can give you 736K instead of the standard 640K for DOS.
- Automatically detects the speed of the memory in your PC and uses fast memory whenever possible.
- Increases system performance by remapping slow ROM into the 386's fast RAM memory.
- Frees up DOS conventional memory by loading memory-resident programs above video buffers.
- Supports 386 protected-mode programs, such as Ansa's Paradox 386, that use Phar Lap Software's 386 DOS Extender.
- Turns DESQview into a 80386 control program, using the 80386's 8086 machine architecture to run multiple large programs concurrently.
- Takes approximately 1.5K of overhead from the PC's conventional memory used to run DOS programs.

System Requirements: 80386 PC, IBM PS/2 Model 80, or standard 8088, 8086, 80286 with 80386 add-in board.

For IBM PS/2 model 50 and 60 users, QEMM-50/60:

- Fully supports the Lotus-Intel-Microsoft Expanded Memory Specification (EMS) version 4.0.
- Takes advantage of the hardware capabilities of the IBM PS/2 80286 Memory Expansion Option to transform its "extended" memory into "expanded" memory.
- On PS/2 Model 50 and 60's with 1.5 MB or more of the IBM Memory Expansion Option memory, enables DESQview to run multiple large programs concurrently.

System Requirements: IBM PS/2 model 50 or 60 with IBM PS/2 80286 Memory Expansion Option board or compatible. Note: IBM PS/2 architecture requires disabling motherboard memory and 'backfilling' conventional memory to 1 MB in order for any multitasking environment to work, including Microsoft Windows 2.0, IBM's 3270 workstation program, or DESQview. QEMM will not run on 80286 computers with extended memory.

Rush Me Quarterdeck Productivity Solutions Today!

Product	No. of Copies	Media 3 1/2" or 5 1/4"	Retail Price, ea.	Total
QEMM/386			$59.95	
QEMM 50/60			$59.95	
DESQview 2.0			$129.95	
Shipping & Handling	USA	$ 5.00		$
Payment method: ☐ Check	Outside USA	$10.00		$
☐ VISA ☐ MC ☐ AMEX	Sales Tax (CA residents)	x 6.5%		$
Valid Since ____ / ____ Expiration ____ / ____			Amount Enclosed	$

Card Number:

Name on Credit Card:

Shipping Address:

City:

State: Zip: Tel:

Mail to address at right.

Quarterdeck

QEMM and DESQview

Quarterdeck Office Systems, 150 Pico Blvd.
Santa Monica, CA 90405 (213) 392-9851

Note: If you own DESQview call us for a special upgrade offer, or send in your DESQview registration card. AST Special Edition owners included.

CIRCLE 90 ON READER SERVICE CARD

by Don Libes

rmifdef

Large numbers of conditional preprocessor directives (*#if, #ifdef, #else*, etc.) can make any code unreadable. This column presents *rmifdef* (Listing 1), a nifty tool to overcome this problem.

Unreadable code due to excessive conditional preprocessor directives is a typical result of code that has been ported several times. There are *#defines* for each environment, and they control which source statements are actually compiled. When *#if*'s and *#ifdef*'s are nested, it can be very difficult to determine whether a statement is used or not, unless you examine many preceding lines and simulate the execution of the preprocessor.

Most C compilers have a flag that allows you to view your source after the preprocessor has executed. This is one way to remove preprocessor directives. Unfortunately, this has several drawbacks. The most severe problem is that macro expansions can make the preprocessed output look unrecognizable from your original source. This is exacerbated by *#include* files that cause the substitution of large quantities of declarations and macros, most of which you don't use. In practice, all that is necessary to make source code readable is to remove a small number of *#define*'s and leave the rest as is.

This column includes the source for a program that does exactly that. It is called *rmifdef* and was written by Sjoerd Mullender of VU Informatica, Amsterdam. Incidentally, Sjoerd was the grand prize winner of the 1984 International Obfuscated C Code Contest (see *Micro/Systems Journal*, September/October 1985).

rmifdef reads C source and a list of defined and undefined symbols. It then outputs the program as if the preprocessor had run *but only processed the directives referencing the symbols given to rmifdef. #define*'s and *#include*'s inside the program are ignored.

Command-line arguments follow the UNIX C compiler conventions. For example, *−Dtoken* behaves as if token was *#define*'d. Similarly, *− Utoken* behaves as if token was *#undef*'d. If any other tokens are encountered, you are prompted interactively on whether those tokens are defined or not. The program is smart enough not to prompt while running as a filter. You can override either of these behaviors. *−a* will force the program to ask you about unknown tokens while *−A* will force it not to ask.

The following program shall be used in all the examples in this column.

```
program.c:
        #ifndef ONE
                a = 1;
        #endif ONE

        #ifdef TWO
                a = 2;
        #ifdef ONE
                a++;
        #endif ONE
        #endif TWO
```

The results of running *rmifdef* on this program differ, depending on the command line. Here is the code if both *ONE* and *TWO* are defined.

```
% rmifdef -DONE -DTWO program.c
        a = 2;
        a++;
```

```
% rmifdef -UONE -DTWO program.c
        a = 1;
        a = 2;
```

Notice that if *TWO* is defined and *ONE* isn't, we get something entirely different. (And this is just a simple program.) This should make clear why it is often hard to read programs with a number of preprocessor directives.

If you don't define or undefine all the symbols, *rmifdef* will ask you about them:

```
% rmifdef -UTWO program.c
is "ONE" defined? n
        a = 1;
```

Sometimes you do not want to remove all the symbols. *−a* will force *rmifdef* not to preprocess lines that you haven't defined or undefined.

. . . continued on page 14

Listing 1. rmifdef

```
/* rmifdef - remove conditional preprocessor directives */
L       [_A-Za-z]
D       [0-9]

%Start SKIP

%{
#define STACKSIZ        100
#define NTAB            1000

#define DEFINED         1
#define UNDEFINED       2
#define UNKNOWN         3

short *sp;
%}

%%

^#[ \t]*ifdef[ \t]+{L}({L}|{D})*        {
        if (*sp & 4 ? (*sp & 2) == 0 : (*sp & 1) == 0)
                *++sp = 0;
        else
                switch (defined()) {
                case DEFINED:   *++sp = 1; break;
                case UNDEFINED: *++sp = 2; break;
                case UNKNOWN:   *++sp = 3; ECHO; break;
                }
        if (*sp != 3)
                BEGIN SKIP;
        }

^#[ \t]*ifndef[ \t]+{L}({L}|{D})*       {
        if (*sp & 4 ? (*sp & 2) == 0 : (*sp & 1) == 0)
                *++sp = 0;
        else
                switch (defined()) {
                case DEFINED:   *++sp = 2; break;
                case UNDEFINED: *++sp = 1; break;
                case UNKNOWN:   *++sp = 3; ECHO; break;
                }
        if (*sp != 3)
                BEGIN SKIP;
    }
```

. . . listing continues

TRY MAGIC PC: THE ULTIMATE DBMS POWER

You know how database applications are created — by hacking out line after line of time-consuming code. Most DBMS' and 4GL's give you some programming power. But when it comes to serious applications, they keep you bolted to your seat writing mountains of tedious code. And rewriting it all over again with every design change.

Imagine how much faster you'd be if you could replace the painful coding phase with an innovative visual technology which takes only a fraction of the time: Introducing Magic PC—the revolutionary Visual Database Language from Aker Corporation:

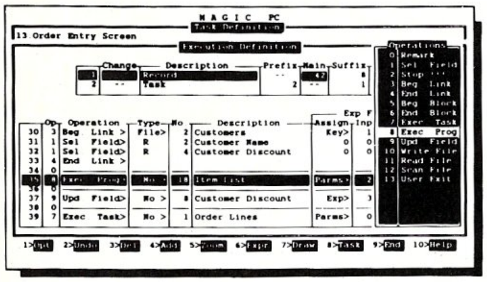
PC TECH JOURNAL

"Magic PC's data base engine delivers powerful applications in a fraction of the time. . . there is truly no competitive product."

Victor Wright — PC Tech Journal

• High-Speed Programming:

With Magic PC's visual design language you quickly describe your programs in non-procedural Execution Tables. They contain compact programming operations which are executed by Magic PC's runtime engine. You fill-in the tables using a visual interface driven by windows and point-and-shoot menus. One table with 50 operations eliminates writing more than 500 traditional lines of code. Yet with Magic PC you don't sacrifice any power or flexibility.



With a powerful set of high-level non-procedural operations you program at only a fraction of the time.

• Maximum Power AND Simplicity:

With Magic PC, you can generate robust DBMS applications including screens, windows, menus, reports, forms, import/export, and much more! Plus, Magic PC has one of the friendliest user interfaces you've ever seen. Using Magic PC you can look-up and transfer data through a powerful Zoom Window system. Magic PC even lets you perform command-free queries.

• Btrieve Performance:

Magic PC incorporates Btrieve, the high-performance file manager from SoftCraft. This gives you exceptional access speed, extended data dictionary capabilities, and automatic file recovery!

• Virtually Maintenance-Free:

With Magic PC you can modify your application design "on the fly" without any manual maintenance. Magic PC automatically updates your programs and data files on-line! This also makes Magic PC an ideal tool for prototyping complete applications in hours instead of days.

• FREE Networking:

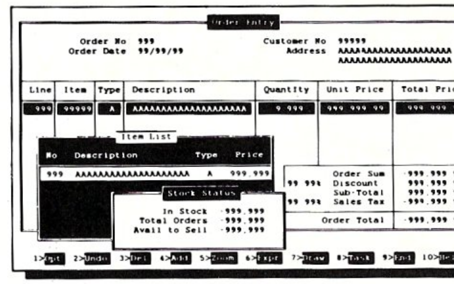
Magic PC comes complete with LAN features. Develop multi-user applications for your LAN with Magic's file and record-locking security levels.

• Stand-Along Runtime:

Distribute your applications and protect your design with Magic PC's low cost runtime engine.

• All For Only $199:

Best of all, Magic PC is an unbeatable bargain. For a limited time, Magic PC's price has been reduced to only $199! Yes, this is the same Magic PC that normally lists for $695! And Magic PC eliminates the need for a separate DBMS, compiler, or application generator. It comes complete with all the tools you need to develop your own database applications instantly.



Pop-up Zoom Windows run multiple programs per screen — with point-and-shoot data transfer between windows!

• $199 — With A Money-Back Guarantee!

For a limited time, you can get Magic PC for only $199. And even at this low price, Magic PC is risk-free. If you're not completely satisfied, simply return it within 30 days and we'll buy it back (less $19.95 restocking fee). And if you'd like a preview, Magic PC's Tutorial Demo is available for just $19.95.

But you'd better hurry — Magic PC's special $199 price won't last long!

• Join The Magic PC Revolution

To unleash your DBMS design power, order your $199 copy of Magic PC right now by calling toll-free or returning the coupon below.

ORDER NOW: CALL (800) 345-MAGIC

In CA (714) 250-1718

MAGIC PC
The Visual Database Language
by AKER

Yes! I want to generate powerful applications much faster!

☐ Rush me my copy of Magic PC at the special promotional price of $199 (add $10 P&H, and tax in CA. International orders add $30). I understand I can return Magic PC for a refund within 30 days, if I'm not completely satisfied.*

☐ Rush me a copy of Magic PC Tutorial Demo at $19.95 (add $5 P&H, and tax in CA. International orders add $15).

Name ____________________

Company ____________ Phone ____________

Street Address (no POB) ____________________

City ____________ State ______ Zip ________

☐ Check enclosed Charge to my: ☐ VISA ☐ MasterCard ☐ AMERICAN EXPRESS

Account No. ____________________

Acct. Name ____________ Exp. Date ___/___

Signature ____________________

Return to: **Aker Corp.,** 18007 Skypark Cir B2, Irvine, CA 92714

SMG

System requirements: IBM PC, XT, AT, PS/2 or 100% compatible with 512K RAM, hard disk and DOS 2.0 or later. 5¼" format, not copy protected. Dealer pricing available. *Return policy valid in US only.

Continued from page 12

```
% rmifdef -a -UTWO program.c
#ifndef ONE
        a = 1;
#endif ONE
```

rmifdef is defined by a lex program. This is appropriate since most of what this program does is to take very simple actions, e.g., echo the input, based on recognizing tokens in the input stream.

> *Macro expansions can make the preprocessed output look unrecognizable.*

Lex will take the following program as input, and produce a C program that you can compile. The output of the C compiler will be *rmifdef.*

Lex is a wonderful tool for writing scanners quickly. Austin Code Works sells a version of lex (with source code) for $25. Versions of lex are also available in the public domain and through the C users group. □

Don Libes is a computer scientist based in the Washington, D.C., area working in artificial intelligence and robot control systems.

All the source code for articles published in Micro/Systems *is available on an MS-DOS disk. To order, send $14.95 to* Micro/Systems Journal, *501 Galveston Drive, Redwood City, CA 94063; or call Tim at (415) 366-3600. Please specify the issue number. Source code is also available on CompuServe; type GO DDJFORUM.*

Did you find this article particularly useful? Circle number 1 on the reader service card.

Product Information

LEX (lexical analyzer generator)

Austin Code Works
11100 Leafwood Lane
Austin, TX 78750-3409
(512) 258-0785

```
^#[ \t]*if[ \t].*\n     {
        if (*sp & 4 ? (*sp & 2) == 0 : (*sp & 1) == 0)
                *++sp = 0;
        else
                switch (true()) {
                case DEFINED:   *++sp = 1; break;
                case UNDEFINED: *++sp = 2; break;
                case UNKNOWN:   *++sp = 3; ECHO; break;
                }
    }

^#[ \t]*else.*\n        {
        if (*sp == 3)
                ECHO;
        *sp |= 4;
    }

^#[ \t]*endif.*\n       {
        if ((*sp & 3) == 3)
                ECHO;
        --sp;
    }

<SKIP>\n          {  BEGIN 0;  }

<SKIP>.           {  /* do nothing */;  }

        if (*sp & 4 ? *sp & 2 : *sp & 1)
                ECHO;
    }

%%

struct table {
        short t_flag;
        char *t_name;
} table[NTAB];

struct table *tabend;
short stack[STACKSIZ];

char *cmd;
int dontask;

main(argc, argv)
register char **argv;
{
        tabend = &table[0]; sp = &stack[0]; *sp = 3;
        cmd = *argv;
        while (--argc > 0) {
                if (**++argv == '-') {
                        switch (*++*argv) {
                        case 'a':
                                dontask++;
                                break;
                        case 'A':
                                dontask = 0;
                                break;
                        case 'd':
                        case 'D':
                                defsym(++*argv);
                                break;
                        case 'u':
                        case 'U':
                                undefsym(++*argv);
                                break;
                        default:
                                error("unknown option");
                                break;
                        }
                } else
                        break;
        }
        if (argc == 0) {
                dontask = 1;
                yylex();
                exit(0);
        }
        while (argc > 0) {
                if ((yyin = fopen(*argv, "r")) == NULL)
                        fprintf(stderr,
                                "%s: cannot open %s\n",
                                cmd, *argv);
                else {
                        yylex();
                        fclose(yyin);
                }
                argv++;
                argc--;
        }
        exit(0);
}

defsym(sym)
char *sym;
{
```

```
        tabend->t_flag = DEFINED;
        tabend->t_name = sym;
        tabend++;
}

undefsym(sym)
char *sym;
{
        tabend->t_flag = UNDEFINED;
        tabend->t_name = sym;
        tabend++;
}

unknownsym(sym)
char *sym;
{
        tabend->t_flag = UNKNOWN;
        tabend->t_name = sym;
        tabend++;
}

defined()
{
        register char *s;
        register struct table *p;

        s = &yytext[yyleng];
        while (*--s > 32)
                ;
        s++;
        for (p = &table[0]; p < tabend; p++)
                if (strcmp(p->t_name, s) == 0)
                        return p->t_flag;
        if (dontask)
                return UNKNOWN;
        return ask(s);
}

ask(sym)
char *sym;
{
        register char *s;
        char buf[128];
        extern char *malloc(), *strcpy();

        fprintf(stderr, "is \"%s\" defined? ", sym);
        s = strcpy(malloc(strlen(sym)+1), sym);
        gets(buf);
        if (buf[0] == 'y' || buf[0] == 'Y') {
                defsym(s);
                return DEFINED;
        } else if (buf[0] == 'n' || buf[0] == 'N') {
                undefsym(s);
                return UNDEFINED;
        } else {
                unknownsym(s);
                return UNKNOWN;
        }
}

true()
{
        register char *s = yytext;
        char buf[128];

        if (dontask)
                return UNKNOWN;
        while (*s++ != 'f')
                ;
        while (*s == ' ' || *s == '\t')
                s++;
        yytext[yyleng - 1] = 0;
        fprintf(stderr, "is \"%s\" true? ", s);
        yytext[yyleng - 1] = '\n';
        gets(buf);
        switch (buf[0]) {
        case 'y':
        case 'Y':
                return DEFINED;
        case 'n':
        case 'N':
                return UNDEFINED;
        default:
                return UNKNOWN;
        }
}

error(s)
char *s;
{
        fprintf(stderr, "%s: %s\n", cmd, s);
        exit(1);
}

yywrap()
{
        return 1;
}
```

End Listing 1

by Stephen Randy Davis

Understanding Units

Many of you will have upgraded to Turbo Pascal 4.0 by now. Some of you may be wondering why, given the problems that can arise when porting working applications from Turbo Pascal 3.0 to 4.0. But take heart. It will be worth the effort once you complete your task.

One of the main justifications for undergoing the inevitable pain of upgrading is the advantage offered by independently linked "units." Programmers who teethed on Turbo Pascal 3.0 may not be familiar with the concept of linking and even weathered old programming salts may never have run across units, so I will take the time to discuss these issues. Before I do, however, my product review basket is starting to run over.

New Turbo Pascal Products

First in the new product pile is P-Tral, a BASIC-to-Pascal translator from Woodchuck Industries. P-Tral is considerably easier to use than other BASIC-to-Pascal translators I have seen. It consists of simply running a single program and answering a few questions. The Turbo Pascal 3.0 code it generates is also somewhat cleaner than others I have seen. However, it does suffer from a problem common to these types of programs.

BASIC programs are not generally as well structured as Pascal programs since BASIC historically does not have as powerful a set of control structures. No matter how good a translator it might be, P-Tral cannot add structure where none exists. Therefore, P-Tral-generated Pascal programs tend to look like BASIC programs written with Pascal symbols, much like a bad translation from one human language to another.

Also, P-Tral does not seem to support the newer Turbo/Quick-BASIC constructs, being restricted to the older BASICA-style programs. However, if you have some BASIC programs you would like to see in Turbo Pascal, give Woodchuck a call. (You might also want to ask them, "How many chucks can a . . . ," never mind).

Next up are two new offerings from Blaise Computing. Blaise has long been in the business of producing professional software support packages for both Pascal and C. Their original support packages for Turbo Pascal, Turbo Power Tools Plus, and Turbo Asynch Plus have now been ported to Turbo Pascal 4.0.

Turbo Power Tools Plus 4.0 consists of some 125 procedures grouped into 14 units that provide support for everything from string manipulation and screen windows right up through menu and memory management to interrupt service routines and terminate-and-stay-resident "PopUp" programs. The somewhat smaller Turbo Asynch Plus 4.0 provides the same support for communications applications and it consists of 41 procedures divided among seven units. Turbo Asynch makes writing serial communication programs a breeze, even those that communicate in the background. Turbo Asynch also includes support for the XModem communication protocol.

Not every one of these procedures is non-trivial, but taken as a whole, both of these packages represent a large body of working code. Of course, Blaise includes source with both packages and provides telephone support. Blaise also supports a BBS that customers can contact. Packages such as these can save software developers a lot of time, which translates to a lot more money saved than was spent for the original purchase price of $129.

Today's Topic

By compiling directly to an executable object program, thereby skipping a link step, past versions of Turbo Pascal were source-code oriented. Therefore, source code for projects of any size became quite large. If programmers broke those projects into modules, the breaks tended to fall around Include files. It was not uncommon to see the following:

```
Program Main;

{$I MODULEA.INC}
{$I MODULEB.INC}
{$I MODULEC.INC}

End.
```

Here, modules A through C represent the real code and Program Main is nothing more than a shell in which to incorporate them.

Fortunately, the Include file concept did allow for the development of support libraries of programs. A programmer might develop a windowing package as an Include file that other programmers could *$I* into their programs during compilation.

There were problems with this approach, however. Even as fast as Turbo 3.0 is, compilations could take some time when every bit of code had to be recompiled every time any part of it changed. Besides, all of the modules being developed had to be ready for compilation at one time. If a programmer were in the middle of editing a module, no other programmer could recompile to test his own changes until the first programmer was finished. This could become quite a developmental nightmare.

But a problem larger than that of human organization was software organization. Large projects must be divisible into smaller pieces that can be assigned to separate programmers for completion. Each of these smaller chunks defines a set of entry points and a function that it is to perform. It is a principal tenet of modern programming that each of these pieces perform its task as independently as possible from the other pieces.

In languages where all source code must be present at all times, this principle can be very difficult to apply. Each programmer sees all of the internals of all other modules. Limiting himself to the specified entry points can take a lot of self control, and yet this is exactly what is necessary to get the job done in a reasonable amount of time.

The solution to these problems is to introduce a separate step between compilation and generation of the executable program, the link step. Modules are compiled independently into what are called object files, typically carrying the extension .OBJ. Modules that contain commonly referenced "library" routines may be converted into the similar .LIB format for quicker processing. Procedures that are referenced from but not defined within a module are left for the linker to resolve as it combines all of the .OBJ and .LIB files together to produce the final executable file.

Unfortunately, the model of independent object modules used by other languages does not fit well with Pascal's strong typing structure. Borland solved this problem by borrowing the concept of a "unit," which main modules can now Include instead of simple *$I* Include files.

A unit looks like any other Turbo Pascal module except for a few addi-

tions. A unit is named using the keyword UNIT in place of PROGRAM. The first section to appear in a unit is the interface section, marked by the keyword INTERFACE. The interface section defines the entry points to the module. It describes in Pascal the module interfaces that were laid out back when the original project was subdivided.

Procedure names are included here along with a description of the number and type of each argument so the result resembles a series of PROCEDURE declarations with no executable statements attached. A unit may contain procedures other than those defined in the interface section, but these procedures must be for internal use only, since they are not accessible by modules that use the unit.

Global variable and type definitions also may be included in this section. This allows a unit to define a new data type or structure for its own use. Having included this unit, other modules may use these new data types or directly reference these global variables. Here, again, a unit may define global variables and data types outside of the interface section, but these are not accessible to units outside of the unit.

The second section is the implementation section marked by the keyword IMPLEMENTATION. This section of the unit contains the actual Pascal source statements. It is this section that corresponds to a standard Turbo Pascal program and that generates the executable machine code corresponding to the .OBJ files of other languages.

The final section is the initialization section. This section is marked by a simple BEGIN and END statement pair and corresponds to the main procedure of a standard program. An initialization section should always be present in every module, but it may be null, that is, contain no statements. The initialization section of each unit is executed after Turbo has initialized its own variables and before executing the first statement in the main program. The addition of this section answers a common problem among library modules—the initialization of data structures prior to use.

Units are included in other programs through use of the keyword USES followed by the name of the units in question. As many unit names as desired may appear, separated by commas. If units themselves use units, they should be declared in the interface section.

Conclusion

We will be making extensive use of units in future columns since their addition is a sizable improvement to Turbo Pascal. In fact, the Virtual Memory Manager unit is already available for downloading from DDJFORUM on CompuServe. New units will be added on a regular basis.

Readers who continue to use Turbo Pascal 3.0 will still be able to profit from units that appear. In general, a unit may be adapted into a regular *$I* Include file by following these steps:

1. Remove the line containing the keyword UNIT
2. Remove any procedure declarations made in the INTERFACE section as well as the keyword itself. (Leave any TYPEs or VARs declared there.)
3. Remove the keyword IMPLEMENTATION. (Leave the procedure definitions appearing in this section.)
4. Assign the initialization section to a procedure name, such as MODULEINIT.

The resulting file can be *$I* included and used in normal Turbo Pascal 3.0 programs. The main program must invoke the initialization procedure before any of the other procedures in the Include file.

However, you will want to start using units as soon as you upgrade to 4.0. As you use them, you will come to understand their value as they increase your programming productivity through additional ease of use and modularity. □

Stephen Randy Davis is a technical editor for Micro/Systems Journal *and a programmer for a defense contractor in Greenville, Texas.*

Did you find this article particularly useful? Circle number 2 on the reader service card.

Product Information

P-Tral *$179*
Woodchuck Industries
340 West 17th (#2B)
New York, NY 10011
(212) 924-0576

Power Tools Plus 4.0 *$129*
Turbo Asynch Plus 4.0 *$129*
Blaise Computing, Inc.
2560 Ninth Street, Ste. 316
Berkeley, CA 94710
(415) 540-5441

Porting DOS Applications to 80386 Protected Mode

by Howard Vigorita

Using Phar Lap's 80386 run-time environment, programmers can adapt DOS programs to run on the 80386.

Programmers who want to make a quick jump into 80386 protected-mode programming need two things: a programming language capable of proper code generation; and an operating system environment in which to run their programs. Phar Lap provides both in its 80386 Assembly Language Development Package, 386/ASM-Link.

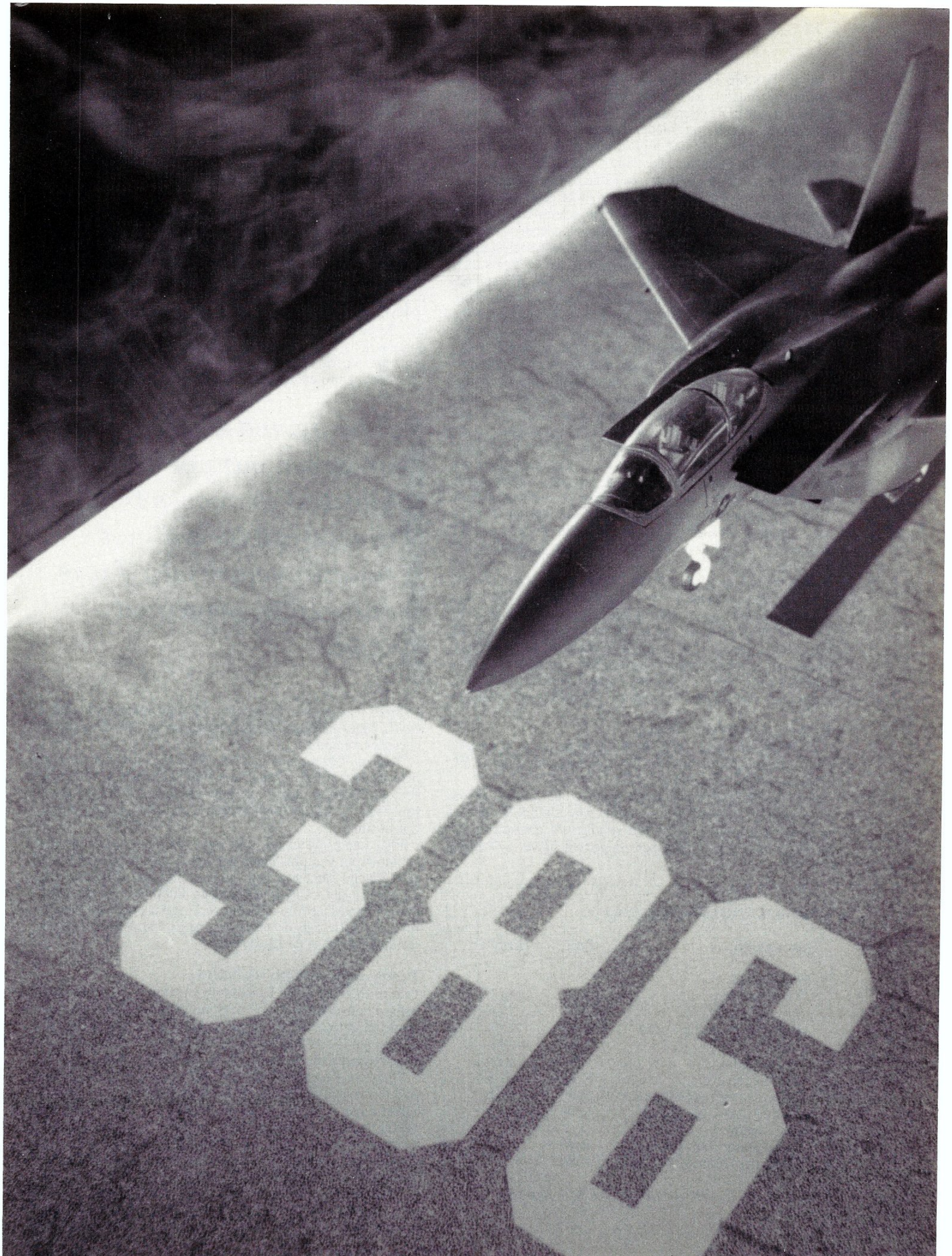
The environment Phar Lap supplies with this package is in the form of a program called RUN386. This tool is meant to be executed on an 80386 machine under MS-DOS. Its function is to load a program into memory and provide the environment in which to execute it.

Phar Lap offers a number of different products that perform a similar function. The MINIBUG debugger supplied with the package does the same thing in a debugging setting. As separate products, Phar Lap also sells 386¦DOS-Extender and BIND386. 386¦DOS-Extender is a memory-resident version of RUN386. It allows protected-mode programs to be run from the DOS command line as if they were regular 8086 programs. BIND386 accomplishes the same thing by linking RUN386 and your program into a single .EXE file.

RUN386 Program Loading

After a program has been created with the Phar Lap assembler and linker, RUN386 can load it into memory and execute it. The syntax to load a program is: *RUN386 <prog> <parms>*. If no file extension is specified, RUN386 will search for a .EXP (protected-mode .EXE) file. Relocatable protected-mode .EXE files (.REX) can also be run if the extension is specified. Older Phar Lap linkers which required that start-up code be explicitly linked, generated a .EXE file. RUN386 is backward-compatible with those .EXE files, but again, the extension must be specified to run them.

RUN386 does not rely on the DOS loader to get a program into memory. Rather, it has its own loader that can load a program as large as available memory.


386

After RUN386 loads a program into memory, it does a number of things before giving over control. It first builds the descriptor tables that the 80386 expects to find when it switches into protected mode. These include an interrupt descriptor table (IDT) and a global descriptor table (GDT). RUN386 also builds a local descriptor table (LDT) in which it defines the segments that are available to your program.

Phar Lap has documented a number of selectors that a loaded program can count on. A valid selector is the only thing that can be loaded into a segment register when in protected mode. Selectors are keys for access to descriptor tables. When a program is given control, the CS register is initialized with a code selector and the DS, SS, FS, and GS registers are all loaded with the same data selector. Note however that the code and data selectors all reference the same physical area of memory. The code selector permits reading and executing while the data selector permits reading and writing. The selectors are summarized in Table 1.

In order to experiment with descriptor table manipulation, I also tracked down a couple of undocumented selectors. A programmer should not count on these selectors not being changed in the future by Phar Lap. Knowledge of the appropriate selector is necessary, however, if a program wants to look at LDT or IDT entries. Although there are privileged instructions for storing descriptor table registers into memory, and table locations can be extracted from these, a program cannot look into just any region of memory. It is first necessary to load a segment register with a selector with read access privileges whose segment spans the table location.

It should be mentioned that you can use the IDT and LDT selectors only because of the way RUN386 initializes your program. The assigned code selector has a value of *0Ch.* Its lowest 2 bits represent the requester privilege level (RPL). The RPL is zero in this case. Inspection of the descriptor referenced by this selector will reveal the descriptor privilege level (DPL) to also be zero. Phar Lap executes the running program in "ring zero," the highest privilege level possible.

RUN386 DOS Emulation

The most important thing RUN386 does is set up a DOS emulation environment. This allows a program to issue *INT 21h* calls from protected mode.

A protected-mode program cannot be allowed to access DOS directly; DOS will not function correctly if it is entered in protected mode. Under the simplest of circumstances, RUN386 must do a number of things to achieve DOS access for I/O:

Table 1. RUN386 Selectors

Description	Type	Number
Documented:		
Program Code	R,X	0Ch
Program DATA	R,W	14h
Program Segment Prefix	R,W	24h
	R,W	04h
Video Buffer	R,W	1Ch
Environment Block	R,W	2Ch
Low Megabyte of Memory	R,W	34h
Weiteck 1167 Memory	R,W	3Ch
Undocumented:		
Local Descriptor Table	R,W	30h
Interrupt Descriptor Table	R,W	50h

R = read, W = write, X = execute

1. Intercept the *INT 21h* software interrupt.
2. Move any extended memory data buffer into conventional memory.
3. Switch to real mode.
4. Reissue the interrupt to DOS.
5. Move any returned data buffer to extended memory.
6. Switch back to protected mode.

RUN386 does not support file control block (FCB) functions. Microsoft has been advising against using FCB functions for years and Phar Lap has taken them seriously. RUN386 does not even create an FCB in the program segment prefix (PSP). In its place it puts three double words of data segment allocation information.

Phar Lap enhances *INT 21h* function pointer parameters for 32-bit addresses. File handle functions and *IOCTL* subfunctions are extended to allow reads and writes of 4-gigabyte blocks. And the memory management functions are redefined to manipulate memory in 4K pages instead of

A protected-mode program cannot be allowed to access DOS directly; DOS will not function correctly if it is entered in protected mode.

16-byte paragraphs. Specifying 4K allocation units with a 32-bit register, Phar Lap is ready now for the 16-terabyte personal computer.

Most of the remaining DOS *INT 21h* functions through *5Ch* are supported. Absent are functions *25h* (set interrupt vector), *31h* (terminate and stay resident), *35h* (get interrupt vector), *4B01h* (load overlay), and *58h* (get/set allocation strategy).

Phar Lap provides six of its own *25h* functions to get and set interrupt vectors. These are not to be confused with the DOS *25h* function, however. The register parameter passing interface is different. The Phar Lap functions provide different combinations for servicing real- and protected-mode interrupts with real or protected-mode handlers. A coding example of this is discussed below.

Programming With RUN386

One of the first tasks I tackled with the Phar Lap package was to convert one of my existing programs to protected-mode operation. The program I chose to convert is the public domain CRC program found on PC/BLUE disks, which is used to check the integrity of the files on the disk.

Before trying to modify CRC for protected mode, I first had to remove all FCB dependencies. Although CRC already made most of its calls via file handle functions, DOS was relied upon to parse the command line. DOS function *29h* (parse file name) was also utilized to parse packed, null-terminated (zstring) file names for display. I had to write these functions myself. That's progress for you.

The next step was to modify the *ASSUME* statements and add segment register set-up code in the vicinity of the code segment declaration. RUN386 does not initialize ES and DS

with a selector for access to the PSP as DOS does.

At this point I decided to make all changes within conditional assembly *IFDEF/ELSE/ENDIF* blocks. Although this tends to clutter things up, it allows the same source code to be assembled for real or protected mode. This also helps to confine bugs induced by changes made to accommodate protected-mode needs.

I also had to do something about the 63.5K disk buffer that CRC uses. In real mode, the stack is moved to the PSP and the last paragraph of the data segment is normalized into a disk buffer segment. You cannot do things like that in protected mode. I decided to use the memory management calls to acquire a proper selector for the disk buffer.

With these few changes the CRC program was up and running in protected mode. But the program ran about 20 percent more slowly than it does in real mode. It did nothing to take advantage of the 386.

The first protected-mode enhancement I had in mind was to take advantage of Phar Lap's extension to the file read function. RUN386 allows a program to read into a contiguous data structure a file block greater than 64K. This involves specifying 32-bit registers as array counters and pointers.

This one alteration pointed up something I had not yet focused on. Until this point I had been referencing registers using their 16-bit designations. In particular, I was clearing a register like this: *Xor CX, CX.* You can get away with this

Table 2. Real- and Protected-Mode Timing Comparisons for XMODEM CRC calculations

	Real	Protected
Disk Read Overhead	7.237	13.209*
CRC Calculation Loop	14.208†	11.724
	13.792‡	

Notes: Times in seconds using external timer hardware
CRC calculated 32 times on 196K RUN386.EXE
on 16 Mhz Compac 386 from VDISK
* includes RUN386 & CRC load time
† read 2-bytes from buffer per loop
‡ read 4-bytes from buffer per loop

as long as the high word of the 32-bit eCX (extended CX) register is clear. But if those high bits ever get contaminated, things can rapidly get out of control.

Consider that the eCX register serves as the counter for string operations and loops. The 80386 doesn't provide distinct 16-bit and 32-bit versions of these operations. If a high bit in eCX gets inadvertently set, a *rep stosb* will do a lot of damage. Similar problems can occur if a contaminated pointer is used to access memory or passed to an *INT 21h* function. Fortunately, the 80386 will issue a segment limit violation error before RUN386 itself gets clobbered.

This problem is easily resolved by replacing all CX references with eCX. The option to assemble for an 8086 is maintained by placing *eCX EQU CX* in a conditional block. For safe measure, I gave most of the other registers the same treatment.

After these changes were made, I found a moderate performance improvement over my first protected-mode effort. I attribute this partly to the fact that the processing loop was able to get more work done with fewer buffer refill interruptions. But the read operation still took 45 percent longer than it did in real mode. This is due to the overhead involved in moving disk buffers from conventional to extended memory with every disk read. I suspect that RUN-386 sometimes moves buffered data when it doesn't need to. I would expect that when a segment is specifically allocated, as the disk buffer was, it should be located in conventional memory if possible. This would allow both the protected-mode program and DOS to access it without shuffling the data.

Performance increases in the CRC processing loop require 32-bit code. Reading the disk buffer 32 bits at a time will speed things up. But the loop control logic must be modified to reflect 4 bytes per loop processing. Essentially, the counter is divided by 2 and the loop internals are replicated. The first pass accesses the 32-bit memory access. In the next pass, the next word is shifted into processing position (see Listing 1). This yields a 21 percent speed increase over real-mode, word-at-a-time processing (see Table 2).

One thing that might catch your eye when looking over the listing is the addressing used. The real-mode CRC loop uses *[BX+DI+256]*(base+index+displacement) addressing instead of the expected *[BX+SI]* (base+index) ad-

dressing to access the second CRC table. The SI index register is already utilized by the string load instruction. In real mode, the only registers that may be used for such addressing are either BX or BP, the base registers, combined with SI or DI, the index registers. The protected-mode loop uses *[BX+BP]*. Protected mode allows any 16-bit or 32-bit general register to function as the base. Any similar register, except the stack pointer, may function as the index.

Intercepting Ctrl-C

The only stumbling block I encountered was intercepting the Ctrl-C interrupt vector. Although full support is now offered, early versions of RUN386 had only a rudimentary function that set real-mode interrupt vectors. This is what first occasioned me to start exploring descriptor tables.

Listing 2 demonstrates how to traverse a descriptor table and locate the Ctrl-C interrupt vector as well as the protected-mode code segment. Also shown is how to locate these things with Phar Lap's new *25h* functions. Function *2505h* is used to replace the real-mode *INT 23h* entry with the vector for the procedure *my-c*. When this new handler gets control, the processor will be in real mode.

> *The only stumbling block I encountered was intercepting the Ctrl-C interrupt vector. This is why I started to explore descriptor tables.*

If all the handler had to do was output a message and terminate, it would be easy. But the reason for intercepting Ctrl-C is to do things like flush buffers or close open files. The handler needs to get back to protected mode to do these things.

Phar Lap's *2506h* function is the answer to this dilemma. It vectors both the real-mode interrupt table and the protected-mode IDT to a single handler. The handler is guaranteed to receive control in protected mode, regardless of which mode the interrupt originated in. This facility can serve as a gateway for real-mode interrupt handlers to transfer control back to protected mode.

An example of this is shown in Listing 2. *INT 21h*, function *2506h* is used to take over user software interrupt *60h* and vector real and protected-mode interrupts to the *exit* process. To enter the exit process, *INT 60h* can be issued from either the real-mode Ctrl-C handler or the protected-mode program. Note that no effort is made in the example to clear the stack or reset RUN386 data structures. This would be necessary if numerous Ctrl-C interrupts could arrive without the program terminating.

Conclusions

RUN386 does a good job of bringing protected-mode operation to existing 80386 based DOS systems. I found the DOS emulation interface complete and well thought out. Atten-

Listing 1. XModem CRC Calculation on the 80386

```
;xmodem CRC via lookup tables. Based on algorithm
;submitted by John Souvestre, New Orleans, LA
;
;DS:eSI  disk buffer pointer
;ES      program data segment
;eCX     disk buffer byte count
;eDX     CRC accumulator
;
         IFNDEF for386
        eAX     equu    AX
        eBX     equ     BX
        eCX     equ     CX
        eDX     equ     DX
        eSI     equ     SI
        eDI     equ     DI
        eBP     equ     BP
        jecxz   equ     jcxz
         ENDIF

_data   SEGMENT DWORD PUBLIC 'DATA'
         IFDEF for386
        odd_byt dd      ?
         ELSE
        odd_byt dw      ?
         ENDIF
_data   ENDS

dta_seg SEGMENT AT 0
         IFDEF for386
dta     db      200*1024 dup(?)
         ELSE
dta     db      (63*1024)+512 dup(?)
         ENDIF
crctbl  db      512 dup(?)
dta_seg ENDS

_text   SEGMENT DWORD PUBLIC 'CODE'
        ASSUME  ds:dta_seg, es:_data, cs:_text, ss:stack

crc_xmdt:
        ; assign registers as follows:
        ;       eBX = word extended input byte
        ;       eDI = pointer to 1st CRC table
        ;       eBP = pointer to 2nd CRC table
        xor     eBX,eBX          ; clear for input
        lea     eDI,crctbl       ; point to 1st table
         IFDEF for386
        mov     eBP,eDI          ; 2nd table 256
        add     eBP,256          ;   bytes away
         ENDIF
        ; divide CX by 4 for 386 or 2 for 8086
        ; and set odd_byt to remainder
        mov     es:odd_byt,eCX ;save full buffer count
         IFDEF for386
        and     CL,0FCh          ;zero out 2 low bits
        sub     es:odd_byt,eCX ;odd bytes are difference
        shr     eCX,2            ;divide byte count down
         ELSE
        and     CL,0FEh          ;zero out low bit
        sub     es:odd_byt,CX  ;odd bytes are difference
        shr     CX,1             ;divide byte count down
         ENDIF
        jz      crc_xmdt_x   ;skip loop if nothing left
        ALIGN 4
crc_xmdt1:

         IFDEF for386

        ; 80386 protected mode code
         IRP x, <1, 2>           ; repeat twice
         IFIDN <x>,<1>           ; 1st time through
        lodsd                    ; get 4 data bytes
         ELSE                    ; 2nd time through
        shr     eAX,16           ; position next word
         ENDIF
        mov     BL,AL            ; turn 1st byte to
        xor     BL,DH            ;   table offset word
        xor     DL,[eBX+eDI]     ; mask table 1 onto CRC
        mov     DH,[eBX+eBP]     ; CRC from table 2
        mov     BL,AH            ; position 2nd byte
        xor     BL,DL            ; table offset word
        xor     DH,[eBX+eDI]     ; mask table 1 onto CRC
        mov     DL,[eBX+eBP]     ; CRC from table 2
         endm
         ELSE

        ; 8086 real mode code
        lodsw                    ; get 2 data bytes
        mov     BL,AL            ; turn 1st byte to
        xor     BL,DH            ;   table offset word
        xor     DL,[BX+DI]       ; mask table 1 onto CRC
        mov     DH,[BX+DI+256]   ; CRC from table 2
```

```
        mov     BL,AH           ; position 2nd byte
        xor     BL,DL           ; table offset word
        xor     DH,[BX+DI]      ; mask table 1 onto CRC
        mov     DL,[BX+DI+256]  ; CRC from table 2

         ENDIF

        loop    crc_xmdt1

crc_xmdt_x:
        mov     eCX,es:odd_byt  ; get count of odd bytes
        jecxz   crc_xmdt_x1

        ; process any odd bytes
crc_xmdt_odd:
        lodsb                   ;get 1 data byte
        mov     BL,AL           ;turn 1st byte to
        xor     BL,DH           ;   table offset word
        or      DL,[eBX+eDI]    ;mask into old CRC
        mov     DH,[eBX+eDI+256];new CRC from table 2
        xchg    DL,DH           ;swap em
        loop    crc_xmdt_odd
crc_xmdt_x1:
        ret

_text   ENDS

stack   SEGMENT DWORD STACK 'STACK'
        db      8*1024 dup(?)
stack   ENDS
        END
```

Listing 2. Exploring 80386 Descriptor Tables

```
;EXPLORE.ASM  Explores Phar Lap's 80386 protected mode
;descriptor tables and demonstrates CTL-C interrupt
;handling and switching between real and protected mode
;

.386p   ; using privileged instructions

        ; Constants
lf      equ     0Ah
cr      equ     0Dh

        ; ease syntax
dwp     equ     dword ptr           ; pointer to 4 bytes
pwp     equ     pword ptr           ; pointer to 6 bytes

_data   SEGMENT DWORD PUBLIC USE32 'DATA'

        ; Messages
signon   db  'Explore 80386 Descriptor Tables ',cr,lf
         db  'by Howard Vigorita, 2/15/88',cr,lf
nl       db  cr,lf,0
c_msg    db  'CTL-C interrupt vector:          ',0
idt_msg  db  'Interrupt descriptor table address: ',0
gdt_msg  db  'Global descriptor table address:    ',0
ldt_msg  db  'Local descriptor table selector:    ',0
ldta_msg db  'Local descriptor table address:     ',0
code_msg db  'Code segment address (hard way):    ',0
csla_msg db  'Code segment linear address:        ',0
cspa_msg db  'Code segment physical address:      ',0
wait_msg db  cr,lf,'Waiting for keystroke (try CTL-C)',0
bye_msg  db  cr,lf,'Leaving Protected Mode',cr,lf,0

        ; storage
        ALIGN 4
dt_limit        dw      ?
dt_base db      6 dup (?)
ldt_selector    dw      ?

_data   ENDS

_stack  SEGMENT BYTE STACK USE32 'STACK'
        db      8192 dup (?)
_stack  ENDS

_text   SEGMENT PARA PUBLIC USE32 'CODE'
        ASSUME  CS:_text, DS:_data

explore PROC    NEAR

        lea     eDX,signon
        call    display

        ; Find address of Interrupt Descriptor Table
        lea     eDX,idt_msg
        call    display
        sidt    pwp dt_limit    ;store IDT reg in mem48
        mov     eAX,dwp dt_base ;display IDT base address
        call    hdout
        lea     eDX,nl
        call    display
```

. . . listing continues

tion has obviously been devoted to many details. Its DOS emulation functions take great care not to contaminate upper register bits. RUN386 will respect VDISK and EMS simulator memory allocations. It even accommodates real-mode terminate-and-stay-resident programs. I was surprised to find that I could pop Sidekick up while a protected-mode program was running.

I found protected-mode programming with RUN386 a valuable learning experience. Given that DOS and most PC hardware limit disk operations to less than 64K blocks, I would not have expected RUN386 to do much better. I wondered if it could do at least as well, though. Taking this into account, I found that the better candidates for protected-mode conversion would be programs that do substantially more data manipulation relative to I/O than the CRC program does.

Programming in the large flat segment architecture of RUN386 was harder for me to adjust to than I had expected. Years of 8086 assembly language programming tends to give one a predilection toward segmented programming solutions. High-level languages, however, are often written to accommodate data structures that span multiple segments. The usual way they do this is by normalizing segment/offset pointer pairs (the segment register is adjusted so that the offset segment never exceeds *0Fh*) with every pointer movement. Code that does this can be radically simplified and will show the greatest performance improvement in 80386 protected mode.

The RUN386 manual, although clear and concise, is obviously too short at 20 pages. The six example programs found on disk are excellent, but I would like to have seen examples covering the more advanced memory allocation, interrupt vector replacement, and *EXEC* functions. Phar Lap is to be commended, however, for its regular technical newsletters and updates.

The solid design and reliable performance of RUN386 make the transition to 80386 protected-mode programming a relatively painless and straightforward process. Assembly language programmers will have little difficulty adjusting to this environment. High-level language compilers will reap the highest performance improvements over large memory model real-mode programs. □

Howard Vigorita is an attorney on the staff of the United States District Court in New York and serves as vice president of the New York Amateur Computer Club.

All the source code for articles published in Micro/Systems *is available on an MS-DOS disk. To order, send $14.95 to* Micro/Systems Journal, *501 Galveston Drive, Redwood City, CA 94063; or call Tim at (415) 366-3600. Please specify the issue number. Source code is also available on CompuServe; type GO DDJFORUM.*

Did you find this article particularly useful?
Circle number 3 on the reader service card.

Product Information

386|ASM-Link *$495*

Phar Lap Software, Inc.
60 Aberdeen Ave.
Cambridge, MA 02138
(617) 661-1510

```
; display IDT entry for CTL-C
lea     eDX,c_msg
call    display
mov     AX,050h         ;GDT selector to IDT data
push    DS              ;save data selector
mov     DS,AX           ;replace it
mov     SI,8*23h        ;point to CTL-C entry
mov     AX,[SI+6]       ;snatch offset
shl     eAX,16
mov     AX,[SI]
mov     BX,[SI+2]        ;snatch selector
pop     DS               ;restore data selector
push    eAX              ;save offset
mov     AX,BX            ;display selector
call    hwout
mov     DL,':'
mov     AH,02h
int     21h
pop     eAX              ; restore offset
call    hdout            ; display offset
lea     eDX,nl
call    display

; Find address of Global Descriptor Table
lea     eDX,gdt_msg
call    display
sgdt    pwp dt_limit     ; 6-byte GDTR to mem48
mov     eAX,dwp dt_base  ; show GDT base address
call    hdout
lea     eDX,nl
call    display

; Find Local Descriptor Table Selector
lea     eDX,ldt_msg
call    display
sldt    ldt_selector     ; LDT selector to mem16
mov     AX,ldt_selector  ; display LTD selector
call    hwout
lea     eDX,nl
call    display

; Find code segment address the hard way
lea     eDX,code_msg
call    display
mov     AX,030h         ;GDT selector to LDT data
push    DS               ;save data selector
mov     DS,AX            ;replace it
mov     SI,8*(0Ch SHR 3);code selector entry
mov     eAX,[SI+2]       ;snatch linear address
rol     eAX,8
mov     AL,[SI+7]
ror     eAX,8
pop     DS              ;restore data selector
call    hdout           ;display code seg address
lea     eDX, nl
call    display

; intercept CTL-C interrupt
; do it easy way with Phar Lap int 21h functions
;
mov     AX,2508h        ;get seg linear base addr
mov     BX,CS           ;code segment selector
int     21h             ;eCX <= linear address
lea     eDX, csla_msg   ;display it
call    display_addr
mov     AX,2509h        ;convert to physical addr
mov     eBX,eCX         ;eBX = linear address
int     21h             ;eCX <= physical address
lea     eDX, cspa_msg   ;display it
call    display_addr
mov     AX,2505h        ;set real mode INT vector
mov     eBX,eCX         ;eBX=linear address
shl     eBX,12          ;cnvt to seg in high word
mov     BX,offset my_c  ;low word=handler offset
mov     CL,23h          ;CTL-C vector number
int     21h

;take over software interrupt for use as gateway
;so CTL-C handler can get back to protected mode
mov     AX,2506h        ;set prot mode INT vector
lea     eDX, exit       ;low word=handler offset
mov     CL,60h          ;user software interrupt
push    DS              ;pass code selector in DS
push    CS
pop     DS
int     21h
pop     DS
lea     eDX, wait_msg ;Ask for a keystroke
call    display
mov     AH,8h
int     21h
int     60h            ;go to exit via interrupt
```

```
explore ENDP

        ; display eDX message and eCX address
display_addr    PROC    NEAR
        push    eCX
        push    eCX
        call    display
        pop     eAX
        call    hdout
        lea     eDX, nl
        call    display
        pop     eCX
        ret
display_addr    ENDP

        ; Print null terminated message to the screen
        ; eDX - Points to the message to be printed out
display PROC    NEAR
        mov     eDI,eDX         ;DI source for scasb
        xor     eAX,eAX         ;search for null in AL
        mov     eCX,eAX         ;set CX as negative
        dec     eCX             ;  down counter
        pushf                   ;save direction flag
        cld                     ;clear it
        repne   scasb           ;scan for null
        popf                    ;restore direction flag
        not     eCX             ;CX now string count
        dec     eCX             ;don't count the null
        mov     AH,40h          ;write device
        mov     eBX,1           ;standard output
        int     21h
        ret
display ENDP

        ; Output hex double word in eAX to the screen
hdout   PROC    NEAR
        push    eAX             ; Output high word
        rol     eAX,16
        call    hwout
        pop     eAX             ; Output low word
        call    hwout
        ret
hdout   ENDP

        ; Output the hex word in AX to the screen
hwout   PROC    NEAR
        mov     BX,AX
        mov     eCX,4           ; once for ea byte
#1:     xor     DL,DL
        shld    DX,BX,4         ; shift hi nib to DL
        shl     BX,4
        add     DL,30h          ; cnvt to ascii
        cmp     DL,39h          ; LT or EQ 9?
        jbe     #2              ; if so, done
        add     DL,07h          ; else, bias to 'A'
#2:     mov     AH,2h           ; output DL byte
        int     21h
        loop    #1              ; again till 4 done
        ret
hwout   ENDP

        ; real mode CTL-C interrupt handler
my_c    PROC    NEAR
        pusha                   ; play it safe
        push    CS
        pop     DS              ; init data segment
        mov     DX,offset ctl_c ; point to message
        mov     ah,09h
        int     21h             ; this is DOS talking
        popa
        int 60h                 ; protected mode exit
ctl_c   db      '<<CTL-C HAS BEEN INTERCEPTED>>$'
my_c    ENDP

        ; exit routine doubles as software interrupt
exit    PROC    NEAR
        lea     eDX, bye_msg    ; say goodbye
        call    display
        mov     AX,04C00h       ; return to DOS
        int     21h
exit    ENDP

_text   ENDS
        END     explore
```

End Listing 2

DMA and Interrupt Driven Real-Time Programming from C and FORTRAN

by D. Curtis Deno

Direct Memory Access and Interrupt Service Routines are examined for data acquisition and real-time control.

This article explores the use of Direct Memory Access (DMA) and Interrupt Service Routines (ISRs) with high-level languages such as FORTRAN and C on the IBM PC/XT/AT family and compatibles. The context of this discussion is data acquisition and real-time control at rates approaching the maximum possible in a multitasking environment coexisting with DOS.

Part 1. Background and Useful Suggestions

There are many technical and scientific applications in which data must be acquired, processed in real time, and made available for subsequent processing. The IBM PC/XT/AT family of microcomputers offers both adequate computing horsepower for many applications and readily available hardware adapters. In addition, the open architecture has fostered good reference material as well as hardware and software.

• ***High-Level Languages***

The consensus is that C is the high-level language most suited for this task, both because the language is richer in data and control structures and because third-party libraries are available to assist with ISRs.

FORTRAN is primarily numeric in orientation and FORTRAN programmers are often unaccustomed to dealing with machine-level specifics, such as programming interrupt or DMA controllers. However, one may wish to utilize routines for signal processing that were developed in FORTRAN or simply have a familiarity that can be used to advantage. With some care, it is possible to have the best of both languages using products that permit mixed-language programming such as Microsoft's C and FORTRAN.

Almost all data acquisition boards (DABs) come with software supporting C as well as FORTRAN. Unfortunately, only rarely can the supplied set of routines do real-time acquisition and control. The primitives supplied frequently have a significant overhead (2–20 ms) and thus cannot simultaneously get analog data, make calculations, and put back out analog signals at rates of more than 10–100 Hz. In spite of this, almost all DABs can acquire data at 20–200 KHz using the manufacturer-supplied routines.

With specially written software, it is often possible to do acquisition and real-time control at rates of 0.5–5 KHz. Whether the high-level language is C or FORTRAN, parts of this special software will probably require assembly language (although this is much less likely in C). The reason for this is the intimate interaction that is necessary with DMA, interrupt, and data acquisition hardware.

• ***Why DMA?***

Since a number of good articles have been written about ISRs and programming the interrupt controller (see the references at the end of this article) this discussion will deal more carefully with DMA. I will try to cover two neglected areas, DMA programming and the use of the 8087/287 numeric coprocessor within an ISR, in addition to reviewing general principles for hardware interrupt programming.

Most high-performance DABs today include DMA hardware that permits high-speed transfer of data into RAM, in addition to interrupt capability at the end of a conversion or a block of DMA'ed conversions. DMA capability is crucial for

maximum speed. During a DMA operation the processor is briefly halted while the address and data buses are manipulated by special-purpose hardware. In contrast, a hardware interrupt requires the CPU to save its internal registers before attending to the event, then requires that transfers go through the CPU, which further degrades performance. The faster DABs available today use the AT's 16-bit bus for high-speed, 16-bit DMA.

• *What's Involved?*

A successful undertaking in real-time software development (assuming functional hardware) will likely require: (1) good references—both hardware and software for computer and data acquisition board; (2) testing hardware—to "single step" generate interrupts, and control and observe A/D, D/A, and binary I/O; and (3) debugging tools—to set breakpoints at source and assembly code lines, to read/modify variables, and to better understand the hardware

• *General Comments on Interrupts*

Two key approaches to control the flow of a program are (1) continuous testing (by the CPU) for the occurrence of some event/flag (this approach is called polling), and (2) interrupts, where (external) events signal their occurrence and force a momentary diversion of the CPU to the ISR. Polling, while simple, is too slow for many real-time applications. Interrupts, however, permit the PC to handle serial data at 9600 and 19,200 baud. Without them, serial communications are restricted to 300 to 1200 baud.

As a general rule, the most time-critical event handler should be interrupt-driven and have high priority. For example, prior to performing an FFT spectral analysis, it is desirable to collect data at a fixed rate. One simple approach is to configure a precise hardware timer to initiate an A/D conversion that will, in turn, issue an interrupt upon completion of the A/D conversion. This interrupt permits the CPU to process this A/D data when it is available. An ISR could then store the data in a user's array and disable the timer when the terminal count is reached. This scheme can work well if the other, higher priority interrupts are brief in duration.

Of primary importance in an ISR is saving the states of the 8086/286 (and 8087/287 if used) prior to using them for the ISR. At the end of the ISR, they are restored completely. Avoid DOS calls inside an ISR because DOS is not re-entrant and the ISR's DOS call may scramble other DOS operations in process (e.g. screen and keyboard I/O). In general, use BIOS calls (most of which are re-entrant) or manipulate the hardware directly even though this is more primitive. You may (and should) take advantage of DOS calls in non-ISR assembly language subroutines, of course. Special steps that may be taken to allow hardware ISRs to use DOS services involve DOS's critical section flag and the interrupt 28 hex DOS idle handler. Not well documented by Microsoft, these approaches are discussed in Blaise Computing's C Tools Plus library.

Keep ISRs short and deactivate other interrupts only during critical sections of code, such as when reprogramming the interrupt controller or resetting the interrupt vector addresses. When the CPU first enters your ISR, only the code segment (CS) is correctly set. It is almost certain you will need a data segment to store intermediate calculations or to gain fast access to useful constants. Defining a new stack segment is probably essential if you use the stack inside the ISR a great deal—although simple ISRs can often freeload off the largess of DOS or the main program. It is worth pointing out that software support of general-purpose ISRs already does most of the above.

One cannot expect interrupts to be serviced immediately, even under the best of circumstances. Dynamic RAM refresh cycles occur briefly and frequently during operation. Approximately every 15 μs, a few clock cycles are used to refresh dynamic RAM. For an 8-MHz AT clone, this is five clock cycles or 625 ns for an overhead of about 4 percent. Higher CPU clock speeds or alternate refresh designs can reduce this overhead further. The main point is, for greatest timing precision, use interrupts to control transfer of data, and use DAB or other hardware to initiate acquisition.

Completely avoid software timing loops. With such a profusion of XTs and ATs with clock frequencies from 4.77 to 20 MHz, variable wait states, and 8088/V20/80826/80386 timing differences, such timing loops are foolhardy. Additionally, they work very poorly in interrupt environments. You may take advantage of the PC's internal clock-timers (not just DOS's clock, which updates every 1/18.2 seconds) to create interrupts at regular intervals from microseconds to seconds, or use the AT's real-time clock (if interrupts at specific times of day are useful). (See Tables 1C and 2C.) It is probably best, however, to leave the system hardware for other concurrent applications (see the section on spawn and system below) and use the special hardware on most DABs.

> *Most high-performance DABs today include DMA hardware that permits high-speed transfer of data into RAM, in addition to interrupt capability at the end of a conversion.*

• *Obey Conventions*

Try to avoid resetting the 8259A interrupt controller's priorities or masking other interrupts. When these are left alone, the main program will be in as natural an environment as possible. Select one of the 8 (PC/XT) or 16 (AT) hardware interrupt request (IRQ) lines (e.g., IRQ 2 in PC/XT or IRQ 9 in AT) that is of sufficient priority and is unused. Special steps need to be taken if an interrupt level is shared (see the *Hardware Interrupt Listing* in the "System Board" section of your *PC/XT/AT Hardware Technical Reference*).

Be aware that the AT gets its extra interrupt levels by chaining a second 8259A into one of the inputs (IRQ2) of the other. This means that two interrupt controllers are in the act. Tables 1A and 2A describe the PC/XT and AT interrupt assignments and may help select the IRQ level.

Avoid configuring the DAB's I/O addresses to coincide with other existing or popular hardware. Since most I/O devices decode only the first 10 bits, I/O addresses of 000H to 3FFH are valid locations (000H through 0FFH are reserved for the system board).

• *Non–ISR Assembly Calls*

It is often very useful to provide small extensions to FORTRAN with short assembly language routines. C users often have such routines already—for example, testing if a keystroke has been detected, reading a serial COM port's settings, or scrolling a portion of the screen (window). Even with FORTRAN, many of these routines are offered by either the compiler company or third party support libraries. However, it is unlikely that all your possible needs have been anticipated. If

you have the skills to write a routine to install and uninstall an ISR, you can certainly write small extensions to FORTRAN or C.

• *High-Level Languages in ISRs*

ISRs can be tricky places to use high-level language routines. The C support libraries help to do precisely this, but precautions must be observed. New with Microsoft C 5.0 is the interrupt attribute for writing ISR functions in C. DOS-mediated I/O (see above) is one potential pitfall. Microsoft and Lattice C both apparently have 8087/287 floating point routines that are not guaranteed to function correctly from within an ISR (see README.DOC file in Blaise Computing's C Tools Plus Version 3.01 for further details). This is one reason why you may want to consider a little assembly language tinkering with an assembly listing of the proposed C ISR. By skirting the C library's 8087/287 routines, custom routines in assembly can guarantee that the numeric coprocessor returns to the main C program in the same state in which it entered.

It is fairly easy to return from an assembly routine without upsetting the stack or registers. When going from an assembly subroutine to a FORTRAN or C subroutine, you must carefully note the steps the compiler takes before it goes to a such a subroutine.

• *Testing and Reentrant ISRs*

Test, test, and retest. Errors, while often dramatically fatal, can be subtle. For example, if the stack is offset after each call to the ISR and the rate of interrupts is low, it may take hours or days for the stack to over- or underflow enough to cause an obvious failure. When interrupts occur at excessively high rates, it is helpful to find out why your ISR fails. By doing so, you may be able to plan a graceful recovery, or, at least, to set flags denoting that an error has occurred.

Other sneaky problems occur if circumstances combine to prevent an ISR from being completed before it is again invoked. A reentrant ISR is code used more than once at the same time. If reentrant ISRs are to be avoided, it may be helpful to use external hardware to gate timing signals. Nested or reentrant hardware ISRs may be useful, however, if one can guarantee that the CPU will catch up later. Steps to promote reentrant ISRs include hiding each invocation's data and stacks, preserving global information, and establishing "busy" flags to forcibly serialize critical sections.

Debuggers are extremely valuable for tracking down errors. Microsoft's CodeView works very well with ISRs if

Table 1A. PC/XT Interrupts

1. Interrupt vector locations are double words (segment and offset) at locations 20–3F hex

Priority	IRQ #	Hex Int #	Usual Function
	NMI		parity & 8087
highest	0	08	timer/clock output 0
	1	09	keyboard
↑	2	0A	rarely EGA & PC Network
|	3	0B	COM2; rarely BSC, Cluster, PC Network, etc.
↓	4	0C	COM1; rarely BSC, SDLC, etc.
	5	0D	fixed disk controller
	6	0E	floppy disk controller
lowest	7	0F	LPT1; rarely IBM's DAB, Cluster, GPIB, etc.

Interrupt controller (one):
8259-5, I/O port addresses 20–21 hex

Table 1B. PC/XT DMA

1. All PC/XT DMA channels have a 20-bit address space (0–1 MB) and a 64K limitation

Priority	DRQ #	Usual Function
highest	0	dynamic RAM refresh
↑	1	rarely SDLC
↓	2	floppy disk
lowest	3	fixed disk

DMA page registers (one):
74LS670, I/O port addresses 80–83 hex
DMA controller (one):
8237A-5, I/O port addresses 00–0F hex

Table 1C. PC/XT Timers

Timer #	Usual Function
0	time-of-day/timebase
1	dynamic RAM refresh
2	audio speaker

Programmable timer (one):
8253-5, I/O port addresses 40–43 hex

Table 2A. AT Interrupts

1. Interrupt vector locations are double words (segment and offset) at locations 20–3F hex (for IRQ0–7) and 1C0–1DF hex (for IRQ8–15)
2. Interrupt levels 8–15 are chained through interrupt level 2
3. On AT, IRQ2 physical line is renamed IRQ9 and software redirection of interrupt occurs so that it is compatible with PC/XT hardware and software

Priority	IRQ #	Hex Int #	Usual Function
	NMI		parity, I/O channel check
highest	0	08	timer output 0
	1	09	keyboard
	2	0A	int controller 2 (chained), IRQs 8–15

Table 2A continued

Priority	IRQ #	Hex Int #	Usual Function
	NMI		**parity, I/O channel check**
	8	70	real-time clock
	9	71	old IRQ2; rarely EGA & PC Network
	10	72	
↑	11	73	
|	12	74	
↓	13	75	80287 coprocessor
	14	76	fixed disk controller
	15	77	
	3	0B	COM2; rarely BSC, Cluster, PC Network, etc.
	4	0C	COM1; rarely BSC, SDLC, etc.
	5	0D	LPT2
	6	0E	floppy disk controller
lowest	7	0F	LPT1; rarely IBM's DAB, Cluster, GPIB, etc.

Interrupt controllers (two):
master 8259-5, IRQs 0–7, I/O port addresses 20–3F hex
slave 8259-5, IRQs 8–15, I/O port addresses A0–BF hex

Table 2B. AT DMA

1. 16-bit AT hard disk "fast I/O" programmed data transfers do not use DMA channels
2. DMA channel 0 is not used for dynamic RAM refresh, as it was on PC/XT
3. All AT DMA channels have a 24-bit address space (0–16 MB) and either a 64K (0–3) or 64 Kword (5–7) limitation

Priority	DRQ #	Bits	Usual Function
highest	0	8	
	1	8	rarely SDLC
↑	2	8	floppy disk
|	3	8	
↓	4		used to cascade DMA channels 0–3
	5	16	
	6	16	
lowest	7	16	

DMA page registers (one):
74LS612, I/O port addresses 80–9F hex
DMA controller (two):
#1 8237A-5, DRQ 0–3, I/O port addresses 00–0F hex
#2 8237A-5, DRQ 4–7. I/O port addresses C0–DF hex

Table 2C. AT Timers

Timer #	Usual Function
0	time-of-day/timebase
1	dynamic RAM refresh
2	audio speaker

Programmable timer (one):
8254-2, I/O port addresses 40–5F hex

Real-time clock (one):
MC146818, I/O port addresses 70–71 hex*

* Programmable as alarm (second resolution) or periodic interrupt (2–8192 Hz in multiples of two)

(1) they are invoked slowly enough, (2) they avoid the nonmaskable interrupt (NMI), and (3) they leave the keyboard's interrupt (IRQ1) alone. Another important use of debuggers is to track and unassemble working code supplied by someone else. This use can help teach the fine points of programming the DAB hardware. Reference books make much more sense when readers can see an example. CodeView easily provides such an example by setting a breakpoint at the source code line where the manufacturer's analogous routine is called. Then, by enabling assembly mode, users can step through and watch the IN and OUT lines for their effects.

• *Compatibility Issues*

If compatibility across the IBM PC/XT/AT family is a consideration, it is usually prudent to write for the lowest common denominator, the 8086 and 8087. Avoid using the 286- or 287-specific instructions that will save a few microseconds. The 80386-specific code in protected mode can yield more significant improvements, and these machines will see increasingly wider use.

Although it is documented in the IBM *AT Technical Reference Manual*, most users may not be aware of the necessity to include a pause between closely spaced I/O port operations decoded by the same I/O chip when using an AT. However, failure to do so in assembly language routines can cause incorrect I/O results. The accepted way to invoke this pause is to insert a JMP SHORT $+2 instruction between closely spaced I/O operations. When unassembled, this looks like a jump to the next instruction (which it is), but it puts enough delay in the instruction prefetch system to allow the I/O decoding hardware to catch up. Pauses are not necessary with a C compiler.

In the future, OS/2 will deserve careful attention and compatibility versus performance tradeoffs may become important. Guidelines and support for well-behaved multitasking routines will be much better developed. A 386 version of OS/2 should be a powerful inducement to programmers to write good real-time software. IBM's new MicroChannel Architecture bus in its PS/2 286 and 386 models is a substantial departure from the past, but it seems to offer significantly better hardware performance.

• *Influences on Main FORTRAN and C Programming*

Assembly language subroutines (ISRs and others) may communicate with their main program by accessing arrays. Error conditions signaled

through flag variables will need to be checked (polled) by the main program. Such an interface permits the orderly resetting of ISR parameters. For example, during concurrent background operation of the ISR, the main program may modify some real-time parameters. When this modification is completed, the main program sets a flag that is checked upon each entrance into the ISR. The ISR, seeing that new parameters are available, gets them and resets the flag. In this scenario, users need not fear catching a partially modified parameter set with the asynchronously invoked ISR.

Be careful to avoid fatal errors in the main program while ISRs are installed. If the error is fatal and returns you to DOS, the ISR is still resident. This can cause the machine to lock up when the next program you run gets some ISR data stuffed into it during an interrupt service, or the ISR routine is replaced by very different code or data. The most common mistakes in FORTRAN are improper I/O so be careful to set ERR= or IOSTAT=. More comprehensive approaches exist, such as adding an ISR disable routine to the compiler's error handlers, and are documented in compiler manuals.

Beware of subscripts or pointers to arrays going out of bounds. Even a compiler's subscript checking can't help an assembly ISR routine. Since data acquisition often involves storing past information in large arrays, circular buffers for data are often established where pointer variables keep track of the buffer's head and tail. If too much data is stored, it overwrites the oldest. Thus, in exchange for more bookkeeping, the data is guaranteed to remain within the array.

- ***Leave Some Room in RAM***

Try not to use up all remaining RAM with your main program and its assembly language routines. If you do leave room, the C routines *spawn* and *system* may be called that let the main program remain resident but load a new copy of DOS. This powerful option is like UNIX's control-Z or the SHELL command in Microsoft interpreted BASIC. With *spawn* and *system* you may check directories, delete or edit files, use the MODE command to alter system settings, and so on. The ISR continues in the background, of course. You return to the main program either automatically or by typing EXIT at the DOS prompt, depending on the details. "Shell escapes" are now common in DOS application software. Such schemes have been used to permit concurrent real-time control and data storage while collection continues uninterrupted.

Part 2. A Case Study

In the following example a programmable digital delay line is needed, and the data available during certain times is to be stored onto disk for later study. This qualifies as real-time control as can be seen from an outline of the process: (1) several analog channels of data are acquired every, say, 0.1–1 KHz and stored in a large circular buffer; (2) binary inputs are read, tested, and stored in the circular buffer; and (3) some past data is found in the circular buffer, scaled by some arbitrary floating point numbers, and put out onto D/A channels. For the remainder of this example I will use specific buffer sizes, sample rates, and so on, to provide a realistic feel for performance and make the discussion easier.

It is useful to have this process operate in the background in order to permit (without breaking the digital delay line or data storage): (1) writing data to disk after certain events;

(2) modifying delay time and scaling factors from the keyboard; (3) querying the program for current delay times, scalings, error flags, and extent of buffer filling; and (4) "shell escaping" to a new copy of DOS in order to edit or delete files.

• *DAB Hardware*

For this example, we will be using the Analog Devices ADI-RTI 815A board which has a maximum of 32 single-ended (16 differential) 12-bit analog inputs, 8-bit DMA (channel 1) and interrupt support [IRQ2(9), IRQ3-7 in order of decreasing priority], two 12-bit D/A converters, 8-bit binary input and output, and three usable 16-bit counter/timers (Advanced Micro Devices Am9513A). Maximum conversion rate is about 50–60 KHz with DMA, gain of 1, and on a single channel. It was purchased with software support (C, FORTRAN, BASIC compiled and interpreted, Pascal, and assembly) and reasonably good documentation. It is not far different from offerings from a number of vendors, including Data Translation and MetraByte.

For higher-performance applications, the new Data Translation DT2821 board offers 16-bit DMA (channels 5, 6, 7), hardware interrupts (levels 10, 15, 3, 5, 7 in order of priority), and 150 or 250 KHz maximum throughput for ATs and compatibles only.

The basic timing interval (0–1 KHz) is to be software-controlled and utilize the 1-MHz oscillator and programmable frequency dividing hardware on the RTI-815A board. For debugging purposes, however, provision must be made to initiate single scans of eight analog channels. The scan is itself a timed progression through the 8 channels and we will use 20 μs/conversion. A complete scan will require 0.16 ms and occur as often as every 1 ms, but this will occur without the CPU's intervention. The hardware is programmed to interrupt upon completion of a scan.

• *Main Program*

A circular buffer size of about 200K to 400K is chosen in order to leave enough room to run small editors and utility programs after having "shell escaped" to a second copy of DOS. It is convenient for FORTRAN, C, and the DMA logic to store data as 1-, 2-, or 4-byte values rather than get clever and pack 12-bit A/D values.

The main program will call routines (C or assembly) to set up the hardware and install the ISR. As discussed previously, the ISR may be some combination of C, FORTRAN, or assembly. The following discussion is based on an assembly language routine for FORTRAN. A number of simplifications can be made to the ISR itself and its installation if a C ISR or support libraries are used.

• *The ISR*

The description of the ISR begins by assuming that the timer has previously initiated a scan of eight analog channels and DMA has placed the resulting data into its appropriate location in the circular buffer. Upon completion, a hardware interrupt is issued.

At this point, the CPU enters the ISR and does the following: (1) turns on interrupts with STI instruction; (2) PUSHes old 8086/286 registers on the stack; (3) defines the default data segment DS appropriately for the ISR's access; and (4) reads the RTI-815's status register to determine: (a) if the interrupt was not from it then disable interrupts and push flags like a real IRQ and call the original ISR, (b) if the interrupt occurred as a result of some error/overrun, then the CPU executes the overrun routine, or (c) if the interrupt occurred at the successful conclusion of a scan, then the CPU executes a normal ISR.

The main part of a normal ISR consists of:

1. Reading the binary input port and storing the result in the appropriate location in the circular buffer adjacent to the A/D data;
2. Testing the binary input for bit patterns that cause flags to be set so the main program can write a portion of the circular buffer to disk;
3. Incrementing a scan number and storing it (or a date/time stamp) in the circular buffer;
4. Getting the old data by subtracting a delay index from the current position pointer in the circular buffer;
5. Saving the registers (state) of the 8087/287 into a predefined location of 47 16-bit words with FSAVE;
6. Doing floating-point computation in the 8087/287 and limiting the results to the valid 12-bit range;
7. Restoring the 8087/287 registers with an FRSTOR instruction followed by an FWAIT;
8. Outputting the delayed and scaled data to the D/A's;
9. Executing the cleanup routine. Steps 5–7 can be simplified if floating-point computations can be safely done within a higher-language ISR.

The cleanup routine at the end of an ISR consists of:

1. Updating buffer pointers;
2. Reprogramming the DMA controller for another scan to the new location;
3. Resetting the RTI-815 for another scan, starting at the appropriate analog channel;
4. Toggling the hardware to permit other pulses to initiate scans;
5. Sending a non-specific end-of-interrupt (EOI) to the 8259A interrupt controller;
6. Popping all the 8086/286's registers back to restore its state prior to entering the ISR; and
7. Executing an IRET to return from the interrupt.

• ***ISR Installation***

The installation of an ISR is not trivial, but since it is reasonably well described in the references at the end of this article, I will outline the steps involved:

1. Get the RTI-815's base I/O address and store it locally for subsequent use.
2. Initialize the scan number counter and other variables needed.
3. Read in the old interrupt mask register (IMR) and save a copy.
4. Activate the desired interrupt level by zeroing that location in the IMR and setting the masks.
5. Use DOS function 35H (software interrupt 21H) to get the current interrupt vector (address) for the selected hardware interrupt level [be careful that IRQ0 is actually interrupt vector location 8H (starting hex address 20) and that IRQ8 is actually vector location 70H (starting hex address 1C0)].
6. Store the old interrupt vector.
7. Use DOS function 25H (software interrupt 21H) to set the interrupt vector to the current CS but with the offset of the actual ISR's start.
8. Set up the RTI-815 for scans.
9. Program the 8237A-5 DMA controller with the starting address and count.
10. Set a flag to indicate successful return.
11. Return to the calling program, popping the appropriate number of arguments off the stack.

• ***Segmentation***

Large circular buffers (> 64K) require 4-byte pointers. When 80386 machines and 386-specific software become widespread, this will be very simple and natural. Unfortunately, the 64K segmentation curse is embodied in 8086/286 machines and almost all software today. Switching back and forth between segmented and absolute addresses is awkward and inefficient, but sometimes necessary.

• ***8087/287 Programming***

There are three major steps in assembly language ISR use of the 8087/287. First, as mentioned above, save the old state with an FSAVE (which will reinitialize the coprocessor to a set of default routines for treating infinity, rounding, and so on). Next, use the 8087/287 to get numbers from memory, perform computations using its internal stack, and return results with appropriate rounding and format conversion. Last, also as mentioned above, the ISR's use of the 8087/287 should end with a FRSTOR and FWAIT in order to restore its previous state and wait for the 8086/286 to catch up.

The numeric coprocessor is very helpful because it not only does trigonometry and floating-point operations rapidly, but it also handles type conversion and rounding. The resulting code is clean and fast, so only rarely will clever integer scaling efforts be any better.

If compatibility across the IBM PC/XT/AT family is a consideration, it is usually prudent to write for the lowest common denominator, the 8086 and 8087.

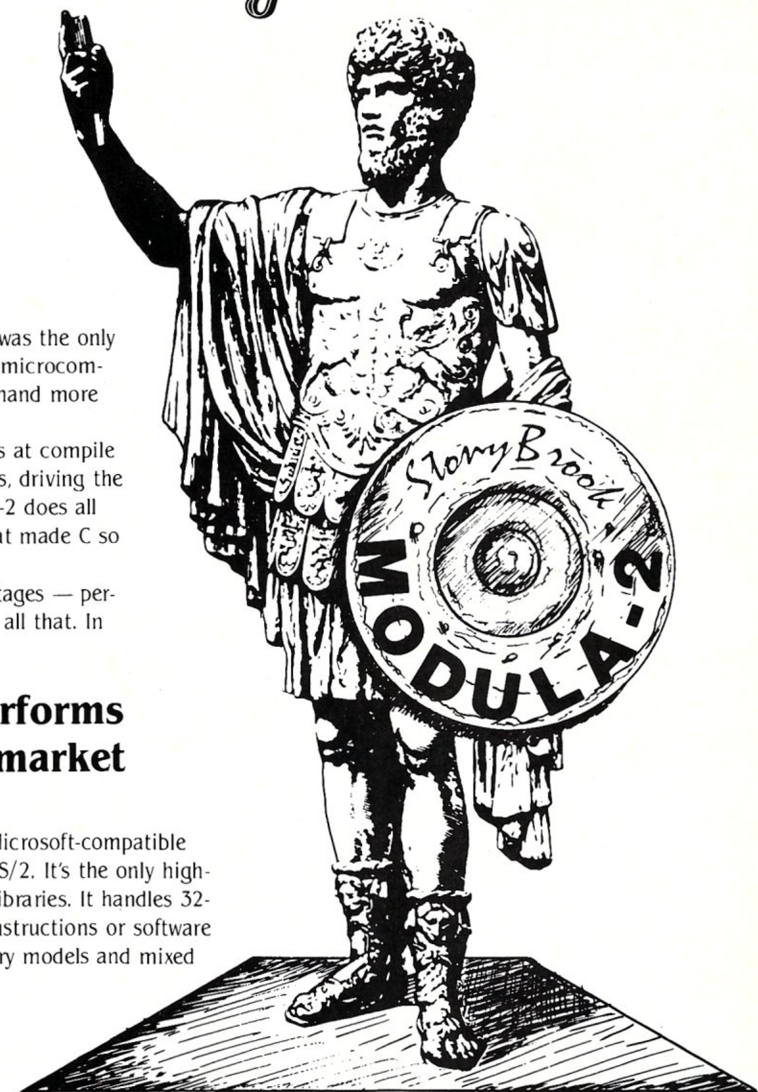
When using Microsoft's MASM assembler (Versions 4.0 and below), remember to supply the .8087 directive and use the /R switch to maintain IEEE standard floating-point format. Beginning with MASM 5.0 the IEEE format is the default. For general programming, the assembler's /E switch permits automatic generation of code that accesses the C or FORTRAN compiler's 8087/287 emulation library. However, as noted in the "High-Level Languages in ISRs" section of this discussion, it may be wise to avoid the emulation code entirely within a real-time ISR.

Another delicate topic in 8087/287 use deals with error handling. Situations such as dividing by zero, loss of precision, or results too large to fit in 2-byte integers are possible. The programmer can either blindly use the supplied result, test a status word, or enable exception interrupt routines. The exception interrupt approach is complex, but it is outlined in C and FORTRAN compiler manuals. In ISRs, a preventative approach may be best.

• ***DMA Programming***

And now, at long last, the subject of DMA programming. DMA, as noted, stands for direct memory access and is used to increase system performance by allowing external devices to directly exchange information with the system memory. Memory-to-memory transfer capabilities are also present. The primary reference for this topic is Intel's description of their 8237A-5 High Performance Programmable DMA Controller which can be found in their *Microsystems Components Handbook, Volume 1.* Also, see the

I have come to bury C, sir, not to praise it.



C served us well in the days of kilobytes and kilohertz. It was the only language we could implement efficiently on our newborn microcomputers. But with today's mega-machines, shouldn't we demand more from our compilers?

Modula-2 increases productivity by catching your errors at compile time. You'll easily modularize and structure your programs, driving the hordes of barbaric bugs into the hinterlands. And Modula-2 does all this without taking away the low-level machine access that made C so popular.

Until now, you had to pay a price for the Modula-2 advantages — performance just didn't measure up to C. But we've changed all that. In a suite of benchmarks developed by PC Week:

Stony Brook Modula-2 outperforms the best C compilers on the market

(and no other Modula-2 compiler even comes close).

Stony Brook Modula-2, for 80x86 machines, produces Microsoft-compatible objects, and fully supports both Microsoft Windows and OS/2. It's the only high-level language compiler that lets you write dynamic link libraries. It handles 32- and 64-bit real numbers with in-line 80x87 coprocessor instructions or software emulation. And Stony Brook Modula-2 supports six memory models and mixed model programming.

You might want to bury your C compiler once you have used Stony Brook Modula-2, but you won't have to. We made it possible to directly call C and other languages from Modula-2, so you won't have to throw away your investment in C code.

So, friends, programmers, and C-users, lend us your ears. Call us or write for more information and to find out how you can get a demo compiler.

Benchmark	Stony Brook M-2 V1	Logitech M-2 V3	Turbo C V1	Microsoft C V4
Magnifier	0.05	0.06	0.05	0.05
For Loop	0.28	0.73	0.44	0.68
While Loop	0.35	0.75	0.49	0.66
Repeat Loop	0.33	0.68	0.38	0.66
Literal Assign	0.33	0.97	0.60	0.94
Memory Access	0.33	1.04	0.55	1.00
Real Arithmetic	4.68	13.40	3.73	2.96
Real Algebra	4.18	12.87	3.07	3.64
Vector	0.77	1.75	1.21	1.26
Equal If	0.66	1.47	0.93	1.32
Unequal If	0.66	1.43	0.99	1.32
No Parameters	0.33	0.60	0.34	1.04
Value	0.50	0.71	0.60	1.21
Reference	0.61	0.77	0.66	1.21
Real Maths	3.90	4.61	7.85	18.51

The compiler package includes DOS runtime library objects and full sources for our split-screen text editor for $195. The development package includes all of the above plus an automatic make utility, a symbolic debugger, and runtime library sources for $345. MasterCard and Visa accepted. Add $5 for shipping in North America, $25 for overseas shipping.

Stony Brook INC. SOFTWARE

Forest Road, Wilton, New Hampshire 03086 ● (603)654-2525 Ask for Cleopatra.

"System Board" section on Direct Memory Access in the IBM PC/XT/AT (Hardware) Technical Reference.

PC/XT's have only one 8237A-5 DMA controller chip. This results in four DMA channels (0–3), which support 8-bit DMA between I/O adapters and RAM in the 0–1 MB range (see Table 1B). Unfortunately, only DMA channel 1 is free for data acquisition unless disk access is eliminated.

The ATs have a superset with an additional 8237A-5 (channels 4–7), but since channel 4 is the cascaded input of channels 0–3, there are actually only three additional DMA channels (5–7) (see Table 2B). Channels 5–7 are 16-bit, and in the AT all channels can access memory anywhere in the 0–16 MB range. Both 8237A-5 controllers have a maximum count of 64K so that either 64K (channels 0–3) or 64 Kwords (channels 5–7) are the most that can be transferred before reprogramming is necessary.

The DMA controller, like the interrupt controller, has a mask register. Like interrupts, the system bus (IBM calls it the I/O Channel) has DMA request lines (e.g., DRQ1) with which an external device may indicate it is prepared to exchange bytes or words. Before programming a channel, it is wise to mask that channel.

There are four sets (one for each channel) of four 16-bit address and count registers. Access to the 8237A-5 is by 8-bit transfer only, and so some ordering is required. It is prudent to clear the byte pointer (first/last) flip-flop before inputting or outputting to these 16-bit registers in order to ensure that the low and high bytes are accessed in the correct sequence. Addresses in the PC/XT (20-bit, 0–1 MB address space) are handled with a 16-bit address (lower) and a 4-bit (upper) page address. In the AT, the page registers are 8-bit, which permits DMA in the 0–16 MB address space.

There is an important distinction in DMA channels 5–7. All DMA memory transfers must occur on even-byte boundaries. When the base address for these channels is programmed, it is the real address divided by 2 that is the data written to the base address register. The reason that 64K or 64 Kword transfers are the maximum is because increments in the word count registers will not carry through the page registers.

A typical DMA reprogramming sequence that might occur in an ISR would look like this:

1. Mask the DMA channel(s) used for A/D.
2. Set the DMA mode register—this depends on the A/D board's requirements but for the ADI RTI-815A this translates into single-transfer mode (rather than block, demand, or cascade), increment addresses (rather than decrement), disable auto-initialization, and perform write transfers (from an I/O device to RAM).
3. Get the desired starting or base address from segmented form into absolute form.
4. Program the 8237A-5 with the low 16-bits and the appropriate page register with the upper 4 (or 8 bits if DMA channels 5–7).
5. Program the 8237A-5 with the (base) byte or word count.
6. Clear the 8237A-5's terminal count information by reading from the DMA status register.
7. Unmask the DMA channel(s).

DMA requests are prioritized (see Tables 1B and 2B). DMA is commonly used in conjunction with interrupts to interrupt at the end of a transfer. Usually the DMA controller will need reprogramming for subsequent transfers. On the ADI RTI-815A board discussed in this example, the system bus's terminal DMA count (T/C) line is monitored, which allows the RTI-815A to set its flags and initiate an interrupt request. Priority of interrupts and DMA is relevant since we are seeking a real-time program to run in the background.

A final note concerning DMA transfer rates. The DMA

controller in a 10-MHz AT operates at the same rate (5 MHz) as in a 5-MHz PC/XT. For these machines, 8- or 16-bit DMA data-transfer bus cycles are five clock cycles or 1.00 μs (actually 1.05 μs in a 4.77-MHz PC/XT). This is the reason why 6- or 8-MHz ATs can have less throughput with a particular 8-bit DAB than a PC/XT. □

D. Curtis Deno is seeking his Ph.D. in electrical engineering at the University of California, Berkeley. He also holds an M.D. degree, and has worked in medical and physiology research. He also does real-time programming projects.

Circle number 4 on the reader service card if you found this useful.

References

Many of these sources provide elaboration, examples, and technical details that can be invaluable when developing real-time software for the IBM PC/XT/AT family.

DOS

DOS 3.x Reference Manual. Comes with MS(PC)-DOS.

DOS 3.x Technical Reference Manual. Essential for access to DOS and system functions; authoritative; extra cost.

Richard Allen King, *The IBM PC-DOS Handbook,* 2d ed. Sybex, 1986. Highly readable introduction to the IBM PC/XT/AT family from a software and hardware overview perspective.

Peter Norton. *The Peter Norton Programmer's Guide to the IBM PC.* Microsoft Press, 1985. Another readable overview for the serious software developer.

BIOS

IBM PC, XT, or AT Technical Reference Manual. Contains essential BIOS listings. Valuable both as examples and for actual use; requires some familiarity with 8086/286 family assembly language.

Peter Norton. *The Peter Norton Programmer's Guide to the IBM PC.* Microsoft Press, 1985. Good BIOS as well as DOS coverage.

PC/XT/AT Hardware

IBM PC or XT or AT Technical Reference Manual. Essential hardware documentation. Has nice section on interrupt handlers including chaining of interrupts. Gives DMA and interrupt channel assignments and the I/O addresses assigned to each. Even if you have a poorly documented clone, this is a must.

Intel Microsystem Components Handbook. 2-volume set, but especially Vol. 1. Definitive reference and very useful applications notes. 8259A interrupt controller, 8237A-5 DMA controller, and other Intel key chips. Available from Intel Literature Sales, P.O. Box 58130, Santa Clara, CA 95052-8130; (800) 548-4725.

Assembler

Microsoft Macro Assembler Manual. (Version 5.0 or later). Version 5.0 now comes in three nicely typeset paperbacks (*Programmer's Guide, Microsoft CodeView and Utilities,* and *Mixed-Language Programming Guide*) and a spiral-bound reference. MASM 5.0 now supports 386/387s and comes with macros and documentation to facilitate mixed-language programming. The new CodeView supports MASM, C, FORTRAN, and BASIC at a source code/symbolic level. It is inadequate for occasional assembly language programmers who will benefit from more introductory documentation (see the following).

IBM Macro Assembler Manual. Both clear and well organized; recommended for beginning and intermediate-level assembly language programmers.

Leo J. Scanlon. *IBM PC & XT Assembly Language.* Brady, 1983. One of a number of reasonably good introductory texts. Later editions include AT and 80286 specifics.

John F. Palmer and Stephen P. Morse. *The 8087 Primer.* John F. Wiley, 1984. 8087 architecture and assembly language interfaces.

FORTRAN Compiler

Microsoft (or other) *FORTRAN Compiler Version 4.0* (3-vol. set). Vol. 1 is the user's guide, with instructions on how to run compiler/linker and assembly/C/Pascal language interfaces. Vol. 2 is the language reference, and Vol. 3 is devoted to CodeView. Since compiler can now optionally create assembly language listings that can be directly assembled, Microsoft urges users to watch how they do things so users can mimic them.

C Compiler

Microsoft (or other) *C Compiler Version 5.0* (3-vol. set). Vol. 1 is the user's guide with instructions on how to run compiler/linker and assembly/FORTRAN/Pascal language interfaces. Vol. 2 is Microsoft (Xenix-compatible) run-time library reference, and Vol. 3 is the C language reference and CodeView. C compiler can also optionally create assembly language listings that can be directly assembled. The C function interrupt attribute is described in the C 5.0 supplemental documentation. Third-party library support is available for ISRs in C.

C Libraries

Blaise Computing, Inc.'s *C Tools Plus, Version 5.0* (for Microsoft C Versions 5.0 and Quick C 1.0). Professionally written set of software tools and documentation. All source code (C and assembly) is included as are examples. Very good section on ISRs (pp. 55-68); other topics include: window, keyboard, string, and screen functions. Available from Blaise Computing, Inc., 2560 Ninth St., Suite 316, Berkeley, CA 94710; (415) 540-5441.

BIOS Error Fix

Richard Norman. "Video function call fix." *PC Tech Journal,* July 1986, p. 41. Software interrupt 10H needs BP register saved before using. Alternatively, DOS itself may be patched as described in this article. This was a problem in earlier PC and XT models.

Interrupt Routines

William J. Claff. "Writing assembly language interrupt routines." *Byte,* 1986 Extra Edition. *Inside the IBM PCs.* Pp. 249-262. Good example of installing/uninstalling ISRs and of good assembly language style.

Chris Dunford. "Interrupts and the IBM PC." *PC Tech Journal.* Part 1: Nov./Dec. 1983, pp. 173–199,; Part 2: Jan. 1984 pp. 143–186. Very good discussion of PC/XT/AT 8259A interrupt controller chip and how to program it.

Paul M. Dunphy. "IBM PC interrupt service routines." *Byte,* Fall 1985. *Inside the IBM PCs.* Pp. 223–227. Although directed toward Turbo Pascal, this is a good supplement to Claff's 1986 *Byte* article.

Blaise Computing, Inc.'s *C Tools Plus, Version 3.01* (see above).

Assembly from FORTRAN

Mark Dahmke. "Using Assembly routines in MS-FORTRAN programs." *Byte,* 1986 Extra Edition. *Inside the IBM PCs.* Pp. 217–228. A good supplement to the Microsoft manual; explains passing variables on stack reasonably well.

Data Acquisition

Peter Aitken. "Passing the lab test." *PC Tech Journal,* Jan. 1984, pp. 75–84, Jan 1984. Although dated, good discussion of data acquisition on the PC family. Does not fit in the more demanding category of real-time control because, although Aitken uses an external rate generator for synchronized data collection, he polls for conversion done and so cannot run concurrently in background.

A/D and D/A Hardware and Software

Choice of a particular DAB and its accompanying documentation and software is one of the most critical of all selections. For applications with strong analog and digital requirements, an adequate all-in-one board may not exist. The usefulness of software will vary with manufacturer. But, even if you'd like to avoid using vendor's software support it is a good idea to get one complete copy of their software and documentation. It is also convenient to get one of their terminal boards for initial connections. If possible, seek source code listings of software support in addition to object modules or libraries.

IBM Documentation

IBM documentation is a valuable source of information for IBM and many IBM-compatible microcomputers. Place orders for books and reference materials directly with IBM. Direct requests for information and orders to: IBM Technical Directory, Box 2009, Racine, WI 53404; (800)426-7282.

80386 Macro Assemblers for DOS

by Howard Vigorita

Programming early microprocessors was a tedious task. Machine language was entered one byte at a time by setting binary switches. No less cumbersome were the memory dumps that were the only debugging tool.

The first assemblers took a big step forward by allowing the programmer to use instruction mnemonics instead of the op codes. In addition, memory locations could be referenced with meaningful names as labels, instead of being referenced by their actual addresses. The assembler would read the assembly language source code program and translate, on a one-for-one basis, each mnemonic and label into binary code. Such source code programs are much easier for the human programmer to read and debug.

Today's modern assemblers go even farther toward reducing the tedium in programming. They provide constructs, called macros, that allow the programmer to replicate data or code sequences, and to define new mnemonics that the assembler can expand into groups of instructions. This review will cover two such macro assemblers: MASM 5.0 from Microsoft and 386 | ASM from Phar Lap.

Microsoft Macro Assembler

The Microsoft Macro Assembler (MASM) is the assembler most widely used on the PC. And it keeps getting better. Since Version 4.0 was last covered here ("Review of Five 8086 Assemblers," *Micro/System Journal*, July/August 1987), Version 5.0 introduced a number of significant extensions and innovations.

• *MASM 80386 Support*

Topmost on the list of extensions for MASM are the new instructions for the 80386. The default instruction set is still that of the 8086, however. To activate the 80386 real mode extensions, the new *.386* directive must be issued. This will enable the assembler to generate instructions that access the 32-bit registers of the 80386. The *.386P* directive is used to allow MASM to emit privileged instructions. These instructions are used by operating systems to manipulate descriptor tables and machine status.

Using a *.386* directive supports the generating of instruction for two types of segments: the *USE16* and *USE32*. If *USE16* is specified in the *SEGMENT* declaration, default 16-bit words will be generated for addresses and word operands. A size prefix byte will be inserted when a 32-bit quantity is required by a *DWORD* specification or by context. The *USE32* segment type specifier is also provided, but it is really only for documentation purposes. Thirty-two-bit segments are the default when a *.386* directive is in effect and no type is specified. Type specification is not legal if a *.386* directive is not in effect.

Where will you use programs with these 80386 capabilities? To begin with, an 80386 processor is required. Next, you will need an operating environment. Unless you are writing one, the operating environment—not the assembler or your program—will provide the segments in which your program runs. DOS deals with only *USE16* segments. The only 80386 environment that Microsoft offers at this time is Windows/386, for which a developer's pack recently has been announced. But, the present thrust of Windows/386 is to take advantage of the 80386 for its own purposes in letting multiple 8086 programs run concurrently. It's hard to believe that Microsoft would be content to provide programmers with MASM's 80386 capabilities simply as a vehicle for moving to a competitive operating system or environment.

• *MASM Segment Simplification*

One of the first barriers that MASM beginners have traditionally had to overcome was segment definition. With MASM 5.0, the *DOSSEG* directive can simplify the defining of segments for .EXE files. The programmer need not name segments, with a *SEGMENT* directive followed by an *ASSUME* directive. Beginning assembly programmers may find this approach easier to use, but the scheme seems to be aimed more at the high-level, mixed-language programmer.

The *DOSSEG* directive causes MASM automatically to assign a default segment name, an order, an alignment, and a combine type to segments declared with the simplified *.CODE*, *.DATA*, and *.STACK* directives. A total of seven kinds of segments are defined.

Before any of the simplified segment declarations may be issued, the programming model must be declared. Programming models are declared with the *.MODEL* directive and may be small, medium, compact, large, or huge. The defaults mentioned above vary depending on the programming model. Although programmers focused exclusively on assembly language may have only limited interest in programming models, those who mix assembly language subroutines with Microsoft high-level languages such as C will profit enormously. Microsoft has standardized its programming model defaults across all of its high-level languages. You need only know the model and the parameter passing interface.

Another default assigned is the size of the address operand for jumps and

calls to, as well as for returns from code procedures. Near procedures are accessed with a 16-bit offset, while far procedures require a 32-bit segment/offset pair. Small and compact models default to near procedures while the rest default to far. But a *PROC* declaration (without the *NEAR* or *FAR* size specifier) is still necessary for these defaults to be activated.

MASM will automatically choose a return instruction (popping either one or two words off the stack) based on the procedure size. New to MASM 5.0 is the ability to specify the return instruction to use. *RETF* will override the normal return of a near procedure, while *RETN* does the opposite in far procedures. These two instructions permit the operator to dispense with procedure declarations.

• *MASM Performance*

Ever since MASM was rewritten from Pascal to C, it has become faster with every release. MASM benefits from every optimization improvement added to the Microsoft C compiler. MASM 5.0 retains its two-pass design, but benefits from a redesign that makes all system memory available for symbol tables. I found that MASM 5.0 showed about a 10-percent improvement in speed over MASM 4.0 when assembling IBM's VDISK. Speed improvements will vary depending on the nature of the source code being assembled.

For those developers requiring mainframe levels of performance, MASM has been ported for cross development on VAX, Sun, and Apollo systems. As of this writing, Version 4.0 is the current release, but Version 5.0 should be available by press time. Microsoft MASM and C cross-development systems are distributed by Oasys.

• *Debugging with MASM*

CodeView is the debugger now shipped with MASM. Since CodeView is such a complex product in its own right, space limitations preclude an in depth review here, but I will give it a brief overview.

CodeView is a symbolic source-code debugger that displays source code, registers, variables, program output, and status information in various windows. It was first released for C language development, and it reveals its heritage in the format in which hex values are entered. CodeView expects hexadecimal numbers to be entered with a C-styled *0x* prefix, instead of the *H* suffix that is more usual in assembler. In addition to the ability to debug with breakpoints, CodeView also provides watchpoints to stop a program depending on changes in variables or memory locations. Although these fea-

tures are a giant step forward in debugging, I still keep Symdeb around to write patches into files. Symdeb is the symbolic debugger that was supplied with the MASM 4.0 package.

MASM has two new options, */ZI* and */ZD,* which put symbolic information into the object files for use by CodeView. LINK has also been enhanced with the */CODEVIEW* option, which puts debugging information into the executable file.

• *Other Changes in MASM*

MASM 5.0 makes a change in the way it implements strong type checking. To accommodate code written for early versions of MASM that weren't as strict, memory variable type mismatches now cause warnings instead of errors. Warnings don't abort the generation of an object file. A new */W* option has been added to either shut off warning messages or enable a new class of lint-style advisory warnings.

The */R* option is no longer needed to enable 8087 instructions and IEEE real number format. It is now the default. A new *.MSFLOAT* directive is provided if Microsoft binary format is needed.

Many programs will display a usage message if the program is run without command-line arguments. When MASM is run without a command line, however, it goes into interactive mode. In interactive mode, MASM asks the user to enter the names of the assembly language source code, the object code, the listing, and the cross-reference files. A new option has been added that gives MASM the ability to output a usage message. The new */H* option will display a help message listing all of the available command-line switch options.

A new *ALIGN* directive has been added that places code or data on any boundary whose address is a multiple of a power of 2. With older versions of MASM, inventive conditional assembly or *ORG* directives were required to accomplish this. The new directive pads out space to the boundary with NOP bytes.

> *Phar Lap has beaten almost everyone to market with its assembly language package for the 80386.*

Other changes and additions include the ability of MASM to use environment variables, to declare communal variables, and to initialize variables defined in the command line.

As this review goes to press, MASM 5.1 has become available. This is an advance over MASM 5.0 compiled and linked with the OS/2 BIND utility. The result is known as a Family App. It will automatically detect whether it is running under DOS or OS/2 and will execute the appropriate operating system calls.

80386 Assembly Language Development Package by Phar Lap

Phar Lap has beaten just about everybody to market with its assembly language package for the 80386 processor. This package includes the 386¦ASM assembler, 386¦LINK, and FASTLINK linkers; MINIBUG debugger; 386LIB librarian; as well as the RUN386 protected mode 80386 runtime environment.

Conceived as an 80386 assembler, 386¦ASM defaults to the 386 nonprivileged instruction set. Segments default to the *USE32* type. The *.8086* directive limits the instruction set to the MASM default of generating *USE16* segments. Syntax for explicitly specifying the segment type is identical to MASM.

80386 support is the prime focus of 386¦ASM. This is the only assembler package that comes with an 80386 protected-mode run-time environment for executing *USE32* code segments from which DOS system calls can be made.

• *386¦ASM Compatibility with MASM*

Most of the existing DOS, assembly language programs were written for some version of MASM. Recognizing this, Phar Lap has sought to design its assembler for MASM compatibility. The goal is to ease moving the existing code over to the 80386.

To aid in specifying the kind of MASM compatibility a programmer might want, the *.COMPAT* directive is supplied. An argument of M1 (the default) specifies Microsoft MASM compatibility as it is documented,

A Fast, MASM-Compatible Assembler

The newest assembler to appear on the DOS scene is OPTASM from SLR Systems. This assembler has no 80386 features, but it is so advanced in design that no assembler discussion would be complete without it.

To begin with, OPTASM is one of the few DOS assemblers that is itself written in assembly language. As a result, it's one of the most compact assemblers around. But don't let the small package fool you. It packs a big punch.

OPTASM combines an assembler and a MAKE utility in one program. MAKE files are specified as the source of assembly file names in a manner similar to the way Phar Lap and Microsoft implement linker response files. Replacing *<file>* with *@<file>* on the assembler command line causes OPTASM to treat *<file>* as MAKE file instead of a source-code file.

Many language packages include a MAKE utility. A MAKE utility can speed the process of assembling programs from multiple source-code modules. It will only regenerate an object module if its source code, or other modules on which it depends, have been touched more recently. An external MAKE will reload the assembler as each module is processed. OPTASM speeds things up considerably when many modules need assembling. It will load itself once, do its own timestamp checking, and process all the modules that need it.

Unlike MASM and 386¦ASM, it is not a two-pass assembler. It is an N-pass processor. It just keeps on chugging until all convoluted references are resolved. This design totally eliminates forward-reference-induced phase errors.

By being able to deal with forward references, OPTASM can do things two-pass assemblers cannot. OPTASM lives up to its name with its ability to change near jumps to short jumps when it detects that the destination is within a 128-byte range. Similarly, segments can be precisely

while M2 supports what Phar Lap calls the "undocumented and questionable features" of pre-Version 5.0 MASM's, such features as assuming a word size in a *mov [DI],imm* instruction if the operand is larger than byte size. MASM 5.0 generates the same error as 386 | ASM does when set to M1.

Phar Lap doesn't try to emulate known MASM bugs. In particular, Phar Lap correctly implements the *LENGTH* operator. Unlike MASM, the = (equal sign) directive reacts the same way as *EQU* when an offset is specified in the operand. The compatibility switches do not affect these.

I found 386 | ASM to be 100 percent compatible with every piece of MASM source code I threw at it with only one exception. A lone error was generated in assembling IBM's VDISK. VDISK is heavily loaded with complex structures. The line that caused the error looks like this: *LDS SI, [BX].DWORD PTR X* (X is a word in the middle of a structure that stores a segment address. It is followed by another word that stores an offset). Strategic placement of parentheses can solve the problem. Writing this statement more conventionally also works: *LDS SI, DWORD PTR [BX].X*.

Phar Lap makes no attempt to imitate MASM at the command-line level. 386 | ASM has no interactive mode. If you run the assembler without telling it the name of the file to assemble, it will not solicit that information. Invoking the assembler without any arguments displays a usage message and a list of switches. The Phar Lap switch scheme bears no resemblance to MASM's. I didn't find getting used to 386 | ASM any problem, however, because I usually run an assembler from a batch file anyway.

One feature Phar Lap lacks is a switch to link segments in alphabetical order. Some older versions of MASM did this by default. Newer MASMs have the */A* switch for backward compatibility with programs that rely on it. An easy way to get around this is to use empty *SEGMENT/ENDS* blocks to declare all your segments at the top of your program. Segments redeclared with the same name will be combined and linked in the order you first declare them. This trick is also useful for ridding a program of forward reference errors without sacrificing data placement control.

An interesting extension that Phar Lap adds is the ability to define symbols that are local to a procedure block. Any symbol that is prefixed with a # will be visible only within the procedure in which it is defined. This relieves the programmer from having to compose label names that are unique across all procedures.

• *Phar Lap Assembler Design*

Like MASM, 386 | ASM is designed as a two-pass assembler. Thus it shares MASM's inability to deal with forward references to variables. This leads to the dreaded PHASE ERROR, in which variable-size assumptions made during the first pass are discovered to be wrong on the second pass. The only way to deal with this problem is to define all data ahead of the code that uses that data.

The Phar Lap development package runs in 8086 real mode, with the exception of FASTLINK and RUN386. It is actually a cross-assembler package capable of 80386 program development on a standard IBM-PC. All of the constituent tools were written with MetaWare High C. MetaWare, in turn, relies on Phar Lap for assembly and for the run-time environment needed to execute the protected-mode code compiled with High C 386.

386 | ASM seems to be written to maximize portability rather than the speed of the assembler. Its speed in assembly is best described as sluggish. Considering that the MetaWare compilers are capable of a high degree of optimization, Phar Lap can be expected to remedy this at will. Higher performance can also be achieved by an 80386 specific compilation, as was done with FASTLINK.

Programming for portability does pay its dividends when code is moved over to a mainframe. Phar Lap has long provided versions of its XA386 cross assembler that run on VAX/VMS and on a number of different UNIX systems. These versions are maintained at the same version level as their DOS counterpart.

The version of Phar Lap's 386 | ASM package that is the basis for this review is Version 1.1v. Phar Lap has not been shy in updating its products to correct bugs and add features. Between March and September of 1987, eight updates were released.

• *Phar Lap Linkers*

Phar Lap ships two linkers with its assembler package. 386 | LINK is an 8086 linker that defaults to linking 80386 object modules. FASTLINK is functionally equivalent, but will only execute on an 80386. Both linkers are capable of linking standard Intel 8086

for DOS That Packs A Real Wallop

combined without inserting needless NOPs. When it does insert NOPs for *ALIGN* directives, it will fill pairs of bytes with the quicker *MOV AX,AX* instruction. OPTASM can also repair out of range conditional jumps by replacing them with an opposite-sensed jump around an unconditional jump. Similar magic is done with loop instructions.

If you think all this capability must take a toll on performance, look at the comparative timings. OPTASM's worst-case assembly time is almost three times faster than MASM's.

SLR has done a good job of identifying most of MASM's bugs, anomalies, questionable features, and shortcomings. In the spirit of compatibility, it will emulate them all. However, a CONFIG program is provided to turn them on and off. CONFIG allows more than 30 settings—such as jump optimization—also to be adjusted in OPTASM.

OPTASM is as MASM-compatible as possible, considering its radically different design. Things that fall by the wayside are Pass 1 listings, as well as calculations that depend on the IF1 conditional directive. In general, OPTASM must resolve IF conditional directives as they are reached and won't allow IFs that do address calculations.

I found OPTASM as source-code-compatible with MASM as 386 | ASM. Just as with Phar Lap's assembler, OPTASM generated one error on VDISK. I had to replace a conditional *ORG* to a 16-byte boundary with an *ALIGN 16* directive. OPTASM so faithfully duplicates MASM, at both the command-line and source-code levels, that Turbo C users who employ in-line assembly code will want to rename it.

With OPTASM, SLR has shown us the way an assembler should be written. This assembler is bound to be a favorite with developers who can't afford to wait for their object files. And I haven't even scratched the surface in exploring the innovative extensions that OPTASM provides.

Object Module Format (OMF-86) object files into DOS standard .EXE files.

For 80386 targets, Phar Lap implements an extension of OMF-86 that it calls Easy OMF-386. This is the default object-file format that is output by 386¦ASM. Easy OMF-386 object files are marked with a special comment header. This object format contains 32-bit segment length, displacement, and offset fields in place of their 16-bit OMF-86 counterparts.

The OMF-386 format used by Phar Lap is different from the format output by MASM. Phar Lap's linkers will not accept MASM output containing *USE32* code segments. Unless Microsoft and Phar Lap decide to support each other's 386 object module formats, it will not be possible to combine MASM's rapid assembly with FASTLINK for 386 development.

The Phar Lap linker output targeted for the 80386 defaults to an .EXP output file. Older Phar Lap linkers (which output an .EXE file) required a startup code to be explicitly linked. This is now done automatically. The linkers can also output Motorola S-record and Intel hex files that are suitable for ROM burners. .REX, relocatable, protected-mode, task-image files also can be output.

Phar Lap's linkers are compatible with MASM output targeted for the 8086. This makes it possible to use FASTLINK to create 8086 .EXE files. Programmers who write modular code will find this quite attractive. As an experiment, I took a short program and began to put each subroutine into a separate file. Microsoft's Version 3.60 LINK outperformed FASTLINK until about eight object files were being linked. By the time 16 subroutines were assembled into their own object files, FASTLINK was completing the job in half the time of LINK; none of the object files linked exceeded 2K. I would expect FASTLINK to show an even greater margin of improvement as the object file size and count increased. Naturally, this would also depend on the system's having a sufficient amount of extended memory.

One feature the Phar Lap linkers lack is overlay support. Most developers who make extensive use of overlays favor the Phoenix PLINK linker.

• *Phar Lap's MINIBUG*

The MINIBUG debugger is supplied with the 386¦ASM package. This a simple debugger that sets up a run-time environment in a manner similar to RUN386 so that protected mode code can be executed under control of the debugger.

The debugger is executed with the following syntax: *MINIBUG <program>*. As with CodeView, the pro-

Table 1. Assembler Comparison

	MASM 5.0	386¦ASM 1.1v	OPTASM .98
Oddities			
.RADIX 16 1D EQ 1Dh	F	F	T *
x db '123',13,10 LENGTH x	1	5	5 *
x dw 5 dup(*†‡) LENGTH x	5	15	15 *
x = offset y mov AX,x	error†	AX <- offset y	error†
Features			
386 support	Y	Y	N
interactive mode	Y	N	Y
Simplified Segments and .MODEL	Y	N	Y
ALIGN	Y	Y	Y
xor AX,imm8	2 bytes	3 bytes	2 bytes‡
/A option	Y	N	Y
local labels in procedures	N	Y	Y

Notes: * Configurable to respond as MASM 5 does
† Switch to ignore offset as MASM 4 does
‡ Configurable for long or short logicals

Table 2. Timing Tests

	MASM 5.0	386¦ASM 1.1v	OPTASM .98
Structures: VDISK.ASM, 2 files, 2304 source lines	3.57	15.76	1.20
Macros: TEST1.ASM, 1 file, 93 source lines, 22,960 lines expanded	39.11	3:48.32	5.87
Large Source File: TEST2.ASM, 1 file, 22,480 source lines	39.76	2:56.15	5.98
Modular Assembly w/MAKE: ADIR105.ASM, 12 files 950 source lines	6.81	11.75	1.65
Link: ADIR105 modules	2.57 *	1.20†	‡

Notes: Timings from RAM disk in MM:SS.nn
Run on 16-MHz Compaq 386 with 2 MB memory
* Microsoft LINK Version 3.60
† Phar Lap FASTLINK Version 1.1v
‡ SLR does not supply a linker

gram name is required on the command line, and no executable files can be either read in or written out while the debugger is in operation. MINIBUG can debug .EXE, .EXP, and .REX files, but not .COM or other files.

In operation, MINIBUG resembles Microsoft's DEBUG. Commands common to both have the same syntax. Added features include commands to redirect I/O to a serial port, assume near or far pointers, and dump/enter memory in ASCII or hex with choice of bytes, words, or double words. Another nice feature MINIBUG provides is execution timing that starts when the G command is entered. 80386 support is provided, not only with 386/387 instruction disassembly, but also with commands to dump descriptor and page tables.

Phar Lap also offers a separate product called 386¦DEBUG. I haven't actually seen this debugger but Phar Lap keeps tantalizing me by including update information in the same technical newsletters that cover its other products. This is apparently a more full-featured debugger, which resembles Microsoft's Symdeb. It supports symbolic debugging with data watchpoints and has a built-in mini-assembler.

• *Phar Lap Documentation*

Phar Lap's documentation consists of a 270-page assembler manual, a 62-page linker manual, a 54-page debugger manual, and a 20-page RUN386 manual, which are supplied together in a three-ring binder. The manual is legible, with an easy-to-read typeface. I found it complete and clear in its descriptions and often refer to it as an adjunct to MASM's manuals. An addition that would be helpful is an appendix laying out known differences between 386¦ASM and MASM.

Conclusions

MASM and 386¦ASM are both highly polished, professional products. MASM, however, has a few well-known oddities. For instance, a *.RADIX 16* directive, which tells the assembler that constants are to be interpreted as hexadecimal, is ineffective on numbers that end in B (interpreted as binary) or D (interpreted as decimal). 386¦ASM responds as MASM does here. Another example is MASM's *LENGTH* operator. It returns incorrect results in a number of instances (see Table 1). 386¦ASM fixes this one. Microsoft has expressed an intention to adjust some of these things in the forthcoming Version 5.1 update.

The most apparent advantage MASM has over 386¦ASM is in assembly time. MASM typically assembled four times faster than 386¦ASM, although the margin narrowed when assembling modular source code files with MAKE (see Table 2). But this could change. Phar Lap's FASTLINK, which runs in 80386 protected mode, outperformed Microsoft's LINK. 386¦ASM is an excellent candidate for similar treatment. My only regret is that I cannot assemble with MASM and link with FASTLINK for Phar Lap's run-time environment. (Those who consider performance more important than 80386 capabilities should see the sidebar.)

At the present time, Microsoft doesn't offer an environment for running the 80386 protected-mode code created with MASM. MASM's 80386 capabilities are geared more toward writing real-mode applications, virtual-8086 (V86) mode device drivers, and protected-mode operating systems and environments.

386¦ASM's strongest appeal will be to those writing application programs for the 80386. The fact that it comes with an environment to run protected-mode code under DOS is unique. The 386¦ASM development package is an accurately documented, reliable performer that will ease the conversion of applications from real-mode to 80386 protected-mode operation. □

Howard Vigorita is an attorney on the staff of the United States District Court in New York and serves as vice president of the New York Amateur Computer Club.

Product Information

MASM, Version 5.0 $150
Microsoft Corporation
16011 N.E. 36th Way
Box 97017
Redmond, WA 98073-9717

386/ASM—386/LINK Software Development Series, Version 1.1v $495
Phar Lap Software, Inc.
60 Aberdeen Ave.
Cambridge, MA 02138
(617) 661-1510

OPTASM, Version .98 $195
SLR Systems
1622 N. Main St.
Butler, PA 16001
(412) 282-0864
(800) 833-3061
BBS: (412) 282-2799

Microsoft Cross C/MASM/LINK $1000 and up
Oasys
230 Second Ave.
P.O. Box 8990
Waltham, MA 02254-8990
(617) 890-7889

ISIS

CP/M **MSDOS**

COMPLETE SOURCE CODE INCLUDED!

ICXPDS: eXchanger now supports the 5 1/4"iPDS format. Manipulation of ISIS-II files using your computer system was never easier.

ICXMDS: Same as ICXPDS, but for MDS 8" systems.

IMXPDS: Reads/Writes 5" iPDS disks on PC's and AT's.

TELEDPLUS: Enhanced serial file transfer program for CP/M, ISIS, or MS-DOS.

ISE: Emulator gives the CP/M and MS-DOS user access to all the ISIS-II languages and utilities.

ACCELER 8/16: CP/M-80 emulator for MS-DOS. Enables PC's to run ISE. (no source code, V-20 incl.)

$89 each
$250 any 3 above

UDI: The 8086 ISIS Emulator runs all UDI applications. $300

ZAS Development Package: Z-8 and Z-8000 Assembler for CP/M, ISIS, and MS-DOS.

Request a catalog of our products!

Copyrights CP/M Digital Research, Inc. ISIS-II and iPDS Intel Corp. MSDOS Microsoft

Western Wares **303-327-4898**
Box C • Norwood, CO 81423

CIRCLE 83 ON READER SERVICE CARD

Conquer Time and Space.

Introducing XO-SHELL™.

Pop-Up Productivity for Programmers.

No matter what language you program in, XO-SHELL will help you hurdle the barriers to working faster and more efficientiy by eliminating programming hassles. Only with RAM-resident XO-SHELL can you:

- DO CROSS-REFERENCING without leaving your editor
- VIEW ANY FILE and TRANSFER ANY SECTION into your editor or to your printer
- VIEW, COPY and ERASE files directly from a SCROLLABLE DIRECTORY DISPLAY
- With a single key stroke RETRIEVE previous DOS commands, then EDIT and REEXECUTE them
- DO SOURCE-LISTING while in your application
- OBTAIN KEY-CODES without a reference and without going through difficult interpretation
- INSERT GRAPHICS CHARACTERS in your source code.

XO-SHELL is for PCs, XTs, ATs, PS/2s, compatibles.

$49

plus $5 shipping & handling

Call today toll-free

(800) 635-5011

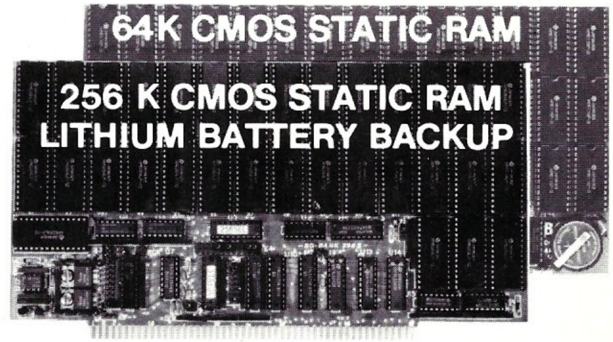
In MA: (617) 868-7704
Visa, MasterCard

WYTE™

WYTE CORPORATION
701 Concord Avenue
Cambridge, MA 02138

CIRCLE 59 ON READER SERVICE CARD

HIGH PERFORMANCE RAM



✓ COMPARE

	8/16 BIT DATA	Bank Selection MPM, OASIS, CROMIX-D	10 Megahertz Speed	Software Write Protect	Battery Backup Option
Compupro Ram 22	✓	NO	✓	NO	NO
Octagon 256K	✓	NO	✓	NO	NO
Cromemco 256KZ II	✓	NO	NO	NO	NO
Dynamic Boards	✓	✓	NO	NO	NO
BG-Bank 256S	✓	✓	✓	✓	✓

GUARANTEED IN YOUR SYSTEM

CROMIX-D • MPM • CCS • OASIS • AMOS

✓**PLUS:** 8/16 BIT TRANSFERS • 24-BIT EX. ADDRESSING
8-12 MHZ • 2K DESELECTS • RAM-EPROM MIX
IEEE 696/S-100 • LOW POWER • FULLY STATIC

LITHIUM BATTERY BACKUP avoids power failure crashes intelligently. Unique POWER-FAIL-SENSE circuit allows processor to save register information and disable board before POWER FAILURE CRASHES memory.

BG BANK 256S........ $329 **Battery Backup**......... $129
BG BANK 64S......... $249 **Battery Backup** $99

MEGABYTE MADNESS
FOUR 256S CARDS
$1199

BG COMPUTER APPLICATIONS, 206 Brookside, Bryan, Texas 77801. International orders add 30%.

(409) 775-5009

CALL ADVERTISER DIRECTLY

Protected Mode FORTRAN Compilers

by Daniel Feenberg, Ph.D.

PC/AT FORTRAN users have long been frustrated by the 640-KB real-mode memory limit. The four FORTRAN compilers reviewed here can all utilize the protected mode of the 286/386 and allow access to arrays up to 16 MB (or the size of physical memory). Three of them use 386 instructions and can offer a speed improvement of two to four times over a real-mode compiler. That gives a PC the calculating performance of a VAX 11/780—substantial by any standard.

Remarkably, this performance is obtained without upgrading to a new operating system. Although running protected-mode programs under DOS seems like magic, all that is required is a run-time monitor to handle the interface between the protected-mode application and the real-mode DOS. The monitor includes a real-mode phase that loads the application into extended memory, and starts it running in protected mode. Interface code receives requests for system services and executes them in real mode via the underlying operating system. As long as the addresses passed in DOS interrupts are valid real-mode addresses, they can be passed to DOS almost without modification. Therefore, system calls can be virtually the same in real and protected mode.

The F77L-EM compiler is a retargeted version of the Lahey real-mode compiler reviewed (and given high marks) in *Micro/Systems Journal* (September/October 1987). It does not use or generate any 386 or 387 instructions, so it runs on any AT with at least 1 MB of extended memory. Of course it will also run on 386 machines. It also differs from the other compilers tested here in that it is not part of a machine-independent family of compilers written in C or PL/I. It is written in assembly code.

The LPI compiler is part of a family of portable compilers offered for a variety of mainframe and minicomputer systems by Language Processors Inc. It is notable for being the first protected-mode FORTRAN compiler offered commercially. As is characteristic of the portable compilers reviewed here, it requires the large linear address space of the 386 processor to execute, and is not available for AT-class machines.

The NDP compiler is a version of the Green Hills FORTRAN compiler common in UNIX environments. It has been modified by Microway to generate 80386/287/387 assembly language code, and the Phar Lap assembler is required for operation. This product does

Table 1. Protected-Mode FORTRAN Compilers At A Glance

	F77L-EM	LPI	NDP	SVS
Version	1.04	02.07.03	1.1	2.7
CPU	286/386	386 only	386 only	386 only
Floating Point	287/387	287/387	287/387/1167	287/387/1167
Minimum extended memory	1 meg	1 meg	1 meg	1 meg
Fixed disk space	.6 meg	2.2 meg	1.4 meg	1.2 meg
DOS required	3.x	3.x	3.x	2.x or 3.x
Linker	Lahey	Phar Lap	Phar Lap	Phar Lap
Run-time system	A.I. Architects	Phar Lap or IGC	Phar Lap	IGC
Compiler price	$695	$795	$595	$895
Additional for linker	included	included	$495	$495
Run-time system	$195	included	included†	included
Total cost	$890	$795	$1090	$1390

† Run-time monitor included with Phar Lap toolset, not compiler.

Table 2. Program Format†

	F77L-EM	LPI	NDP	SVS
filename extension	.for	any	.f	.for
free format	i-stand	i-stand	i-stand	i-stand
include	i-stand	i-stand	i-stand	i-stand
do while	no	no	i-stand	i-stand
trailing comments	i-stand	i-stand	i-stand	no
conditional compilation	i-stand	no	i-stand	i-stand
lowercase accepted	i-stand	i-stand	i-stand	i-stand
& for continuation	yes	yes	yes	yes
* for comment	yes	yes	yes	yes
' and " for delimiters	yes	yes	yes	yes
$ and ⅓ in names	yes	yes	yes	yes
characters allowed in name	31	32	31	31

† i-stand means feature is implemented according to MIL-STD-1753, ANSI/ISA-S61.1, or simply conforms to widespread industry practice.

KEEP UP WITH THE OS/'s

Megabytes of Memory and 32-Bit Performance for DOS.

If you thought the only way to protected mode was the big move to OS/2...We have good news! You can gain the benefits of protected mode the easy way with OS/286™ and OS/386™. These tools for C, Fortran, Pascal and assembly language programmers permit rapid conversion of existing DOS applications from "real" 8086 mode to "protected" 286 and 386 mode. They don't replace or modify DOS, but extend it to protected mode. This way you get multi-megabytes of directly addressable memory (16Mb-286, 4Gb-386) with the compiler, TSRs, device drivers, graphic routines, etc. you use today.

OS/286 and OS/386 are the only DOS extenders that span both the 286 and 386 processors. They run on the widest array of AT and 386 machines, with 32-bit capability *today* on 386s that yields twice the performance of 16-bit mode.

OS/286 and OS/386 are the preferred solutions for developers of high performance memory-intensive applications, including CADKEY, CASE, and Gold Hill, and premier language developers Lahey and MetaWare.

Our optional TOUCHDOWN™ BIOS supplement provides fast and reliable protected mode operation on *any* 286 system, even those with problems resetting the 286. (Ever notice how few existing machines Operating System/2 runs on?) TOUCHDOWN is not required for most major brand AT clones, but for the older machines it is a lifesaver!

If your applications are running out of memory or need more speed, enhance them now without abandoning your investment in DOS.

Coming Soon
WINDOWS 2.0
and DESQview 2.0 API support

Give your protected mode DOS a powerful presentation layer like OS/2, SAA or DESQview.

Special $49.50 Evaluation Offer

Check out for yourself the benefits of protected mode. Our $49.50 "sampler special" is a complete OS/286 Developer's Kit, but with a time limited, non-distributable kernel. There's no better way to learn about the outstanding features of OS/286 and OS/386 than to try them. Of course, the $49.50 is applicable to the purchase of the full OS/286 (16-bit) or OS/386 (32-bit) kit at $495 for either one.

The OS/x86 Developer's Kits include support for popular, C, Fortran, Pascal, Lisp, Prolog, compilers, and assemblers from: MetaWare, Lahey, Microsoft, Lattice, Gold Hill, LogicWare and Phar Lap. A number of other packages are also supported including PLINK86, Halo & GSS Graphics, DESQview and soon Windows 2.0.

Run time licenses for OS/286 and OS/386 are available from $40/copy to under $1.00.

The HummingBoard® turns any XT, AT or 386 into the fastest, most capable 386 system available. The dual processor architecture provides up to 900K each for multiple real mode applications (Lotus, etc.) which can co-reside with protected mode applications on the HummingBoard's 20MHz 386, while debuggers, editors, networks run concurrently on the base processor. 1-24MB memory, 8,870 Dhrystones. A MUST TRY!

A.I. Architects, Inc.
One Kendall Square, Cambridge, MA 02139
TEL (617) 577-8052 FAX (617) 577-9774
Credit card orders only – 24 hours. 617-577-1305

extensive optimization. Weitek support is included in the package, but "has not yet been extensively tested," according to Microway sources.

SVS FORTRAN-386 is a retargeted version of the famous 68000 compiler from Silicon Valley Systems. Its general specifications are similar to the NDP compiler, but SVS generates 386 object code directly. It also includes support for the Weitek 1167 chip set. SVS provides the only source-level debugger among the 386 products.

The required linkers and run-time monitors come from a variety of sources and are not always included with the compilers. NDP and SVS don't include any linker, but specify the Phar Lap product. LPI includes a free copy of the same linker, while Lahey includes a copy of its own linker. Table 1 shows a similar story for the run-time monitors. The Phar Lap product is the pioneer here, and supports both LPI and NDP. The A.I. Architects monitor supports the 286 CPU and is the natural choice for Lahey. The monitor chosen by SVS is from Intelligent Graphics Corp. Linked executable files from the LPI compiler can also run with the IGC monitor. This is of particular interest because IGC claims that programs running under its monitor will also execute under VM/386, the company's multitasking OS. If you already have a protected-mode compiler, you may be able to avoid purchasing another linker or run-time monitor, but only if you already have the required brand.

Among these products only LPI is officially certified by the Federal Software Testing Center. While I feel this is a valuable indicator of quality, it is less than a guarantee, both because the tests are not comprehensive and because it is possible to be certified with errors. The greatest value of certification is that it cannot be renewed unless all previously found errors are corrected. It is also costly, and for this reason some developers have foregone it. SVS claims to have run all the tests in-house, without applying for certification.

In addition to the ANSI 1978 standard, there are two other important standards: the ANSI Industrial Real Time Standard and the Military Standard. Although all these compilers support many features of these newer standards, none claim complete compliance, and none has even attempted to comply with the proposed FORTRAN 8x standard.

Input/Output

Input/output is the area that has defied standardization most truculently. And that trend continues with the compilers we're considering here. Three of these accept units 5 and 6 for preconnected console I/O, but one (SVS) opens unit 0 instead. F77L-EM and NDP preconnect 0 for standard error (not redirectable at the DOS level), but SVS uses 1 for that function. F77L-EM accepts the *namelist* statement, but I consider this a compatibility option to be avoided.

The ability to append output is present only in F77L-EM and LPI. This is a serious omission in NDP and SVS, as is the absence of a binary read in NDP. Although you can read a non-delimited file with unformatted direct access, the process is tedious and presumes you know the length of the file. This means that you couldn't write a program in NDP FORTRAN that could read a Lotus worksheet. Nor could you read binary files produced by real-mode compilers. While LPI and Lahey have chosen unique extensions for this ability, SVS has copied the syntax of the Microsoft FORTRAN compiler (*form= 'binary'*) for this extension.

SVS has an additional problem: a 512-byte limit on formatted records. This is not a violation of the standard, but it's likely to cause problems. The work-around (use binary records and process the delimiters explicitly) is a minor annoyance. The first copy of the LPI compiler I received truncated to zero length any file opened with *status= 'unknown'* or with status unspecified. While not a violation of the standard, this is not what most people

expect, and I was pleased to learn that this behavior will be modified in the next release.

I should mention one dubious extension in the LPI compiler that I liked. On free-format writes, if a numeric follows a character variable, no leading spaces are inserted before the digits. So it is very easy to produce output like "$999." (without spaces after the dollar sign) even if the number of significant digits is not known at compile time. What makes this a feature instead of a "feature" is that it improves the operation of the program without requiring nonstandard syntax in the source code.

Type and Initialization

Table 3C shows some of the extensions these compilers offer. Type specifiers such as *integer*2* and *complex*16* are so universal that many

Most compilers now offer some mechanism for flagging undeclared variables.

FORTRAN programmers think they are part of the standard. I was surprised that double complex was omitted from LPI and SVS. Other extensions, such as *integer*1* are rare but potentially usable. The SVS manual claims that 1- or 2-byte integers can't be used as function arguments. This doesn't seem to be the case but I was unable to determine the nature of the actual constraint. SVS tech support warned only against using those types as returned values in an *INQUIRE* statement. Z format for hex constants is almost universal now, and all these compilers support it. The maximum size of a character variable should be 32,767 or better to be safe. The limit of 4,096 characters in SVS will cause trouble for some users.

Most compilers now offer some mechanism for flagging undeclared variables. I prefer to see this as a compiler switch, so that nonstandard code isn't embedded in the program, as it will be if "implicit none" or "implicit undefined" statements are used. There is the double advantage that the switch can be embedded in a .BAT file and will not depend on the programmer's memory.

The ANSI standard doesn't specify how uninitalized common blocks or local arrays are to be allocated. Whether allocated statically or dynamically, they should not take up disk space. However, only LPI and SVS achieve this. With the NDP compiler, the AUTOMATIC keyword is available to specify stack allocation of local arrays, but this requires introducing a nonstandard syntax where none is required, and does not provide run-time allocation of common storage.

A related characteristic of the standard that is likely to be a continuing source of trouble is the failure to allow array sizes to be determined at run time. Any program that is even moderately general purpose is going to operate on different-sized data sets. For this reason, it should allocate memory at run time, because it is only then that the amount of data is known. For a real-memory machine like the PC, you can pretty much assume everyone has at least 640K and size arrays accordingly, but for protected-mode compilers it isn't so easy. Every additional megabyte you allocate at compile time reduces the number of machines that can run the program, but if too little space is allocated, then users can't use any additional memory that they may have. Each of these

Table 3. Selected Extensions

	F77L-EM	LPI	NDP	SVS
A. Input/Output				
access='append'	yes	yes	no	no
hex formats (Zn)	except character	integers only	yes	yes
binary stream I/O	unique	unique	no	i-stand
preconnected units	0,5,6	5,6	0,5,6	0,1
suppress carriage return	unique	i-stand	i-stand	i-stand
I/O unit range	0–32,767	0–99	0–255	0–99
namelist	yes	no	no	no
B. Functions and Subroutines				
break detection	yes	no	no	no
date and time	yes	no	no	i-stand
get command line	yes	no	no	no
execute subshell	yes	no	no	no
get environment	no	no	no	yes
random numbers	yes	no	no	yes
ior,iand,inot,ieor	yes	yes	yes	yes
ibset,ibclr,ibtest	no	no	yes	yes
C. Type and Initialization Extensions				
integer*2,*4	yes	yes	yes	yes†
integer*2 switch	yes	yes	yes	yes†
integer*1	no	no	yes	yes†
complex*16	yes	no	yes	no
flag undeclared variables	implicit or switch	implicit or switch	implicit or switch	implicit
logical*1,*2,*4	except *2	yes	yes	yes
hex constants	i-stand	i-stand	i-stand	i-stand
maximum size of a string	32,768	32,767	65,535	4,096
mix character and numeric	yes	no	no	yes

† 1- and 2-byte integers partially supported (see text).

Table 4.† Performance Benchmarks

	F77L-EM	LPI	NDP	SVS
A. Whetstone Benchmark				
Double Precision				
compile 225 lines†	5.94s	21.91	40.35	10.76
link	8.52	29.39	—	9.67
execute	17.91	15.27	8.45	11.31
executable size	31.7K	101K	86.2K	38.9
KWi/sec				
(287 Instruction set)	456.8	565.1	485.5	544.2
(387 Instruction set)	—	857.1	1150.2	1170.
Single Precision				
KWi/sec				
(287 Instruction set)	485.5	686.3	504.2	799.1
(387 Instruction set)	—	1048.4	1260.9	1379.5
B. Linpack Benchmark				
Double Precision				
compile 770 lines	13.5s	53.2	114.	19.06
link	24.83	29.1	—	10.0
execute	456.	152.	150	150
executable file size	687K	110K	739K	42K
mflops	.045	.133	.138	.139
Single Precision				
executable file size	364K	110K	416K	41K
mflops	.048	.184	.176	.181

compilers contains in its run-time library a subroutine (usually called *malloc*) that is capable of allocating memory at run-time. Unfortunately, you will need to buy a protected-mode C compiler to access it.

System Interface and Other Functions

A program of professional quality needs access to system resources such as the date and time and contents of the command line, and it needs to recover from (not ignore) keyboard breaks and so on. The writers of these compilers seem to agree with this, because the compilers themselves utilize these services. Apart from Lahey, however, the compiler authors apparently didn't think users might plan to write packages of high quality, because they didn't provide access to these system services. You can provide these yourself with a dose of assembler or C, but that can be expensive in time and money. Although each of these compilers has an associated C compiler through which interlanguage calls are possible, only Lahey allows arguments to be passed by value from the invoking FORTRAN program. With the other compilers, the C side of the call must be specially written to receive all arguments by reference. This also implies that you will never be able to buy a compiled library of C functions and call them from your FORTRAN program without writing special C procedures to make the transition.

Benchmarks

The benchmarks offered here are simple ones. The sieve test determines array-indexing ability. The Whetstone and Linpack benchmarks both attempt to be typical FORTRAN loads, but the Whetstone includes only small one-dimensional arrays and the Linpack emphasizes large (200 by 200) arrays. I have presented single- and double-precision results separately for the floating point benchmarks.

A single benchmark reveals little about any serious performance issue, but an examination of the timings in Table 4 does tell a coherent story. First, look at the compile and link times. These are crucial for any development effort and often exhibit wide variation. In this case, the SVS and Lahey compilers post speeds better than twice the others. The short link time for the SVS compiler is remarkable since all three 386 compilers use the same linker. The explanation no doubt lies in the treatment of uninitialized arrays.

Run times are critical once a program is fully debugged, and here the 286 instruction set greatly handicaps

the Lahey compiler. (Of course, if you have only an AT, the Lahey compiler runs infinitely faster than the others.) The 386 compilers offer about double Lahey's throughput. With the 387 instruction set the SVS compiler wins both floating-point tests, but LPI is the winner when only the 287 instruction set is used.

The sieve test is one of array-indexing ability. Aside from some integer addition the sieve does no arithmetic at all. It was run once for each compiler with a 32K array of potential primes, and again with a 1-MB array. In both cases, 8,191 elements were searched. The not-too-surprising results here show that the 32-bit compilers have the same high performance independent of array size, whereas the Lahey compiler slows down considerably when dealing with arrays larger than 64K. Of course, the sieve represents an extreme example because it does so little else.

Table 4 continued

C. Sieve Test				
(32K array)				
compile 22 lines	2.92s	11.3	25.92	9.28
link	9.72	28.17	—	8.95
execute	5.27	4.51	3.35	3.07
executable file size	56K	98K	115K	28.1K
seconds/iteration	.077	.048	.0242	.032
(1-MB array)				
compile	2.91	11.4	26.14	9.55
link	31.3	28.2	—	9.01
execute	15.6	4.8	3.3	2.96
executable file size	1073K	98K	115K	28.1K
seconds/iteration	.137	.048	.024	.033 D.
Input/Output Tests‡				
compile 27 lines	3.02s	12.3	31.03	9.39
link	9.23	31.0	—	10.77
character write to console	23.1	23.1	26.3	42.3
character write to disk	2.7	3.9	4.4	3.3
unformatted write to disk	2.3	2.91	2.8	2.3
unformatted read from disk	2.3	1.54	1.54	1.3
read character and convert to float	9.9	9.4	6.0	4.9

† Times in seconds on 16-MHz 386/387 Intel motherboard. Tests labeled "287 instruction set" are timed with 387. Compile times include all steps prior to linker, except that for NDP only, link time is also included. File size is for linker output and excludes run-time monitor.
‡ Time in seconds to read or write 1,000 40-byte records. I/O list is single-character*40 variable or 8-element real array.

If you plan to distribute the output of these compilers, you immediately run into three distinct problems.

The I/O benchmarks show times to read or write 1,000 records in various ways. The uniformity of performance here is remarkable. Users might be concerned that the overhead of switching back and forth between protected and real mode to process I/O requests will slow execution down. The Lahey real-mode compiler offers an opportunity to test this. Surprisingly, the timings are essentially the same for both real (not reported here) and protected modes.

Distributing Executable Code

If you plan on distributing the output of these compilers, you immediately run into at least three serious problems. The easiest is the legal requirement that each of your users have a license for the appropriate run-time executive. A few of your customers may already have purchased any one of the C, Pascal, or FORTRAN compilers that

come with the appropriate run-time monitor, but they will form a very small market. Rather more practical is to obtain a binder program that combines the run-time package with the protected-mode executable file to form a stand-alone package that loads itself into protected mode and executes there. These are available from Lahey for their compiler, at a cost of $40 per user in packages of 10. The NDP and LPI compilers use a similar package from Phar Lap that costs $1,495, but there is no per unit royalty. The SVS compiler already comes with a royalty-free license for an equivalent package.

The second problem is that object code from the 386 compilers suffers from a severe lack of portability. There are few enough 386/287 machines out there; the number with 387 (or even 1167) floating-point units is much smaller. You can either distribute a 386/287 version for everyone, or sell separate versions for 287 and 387. Those three versions are only the start, though, because you will almost certainly need a 286/287 version for that much larger market, and then an OS/2 version, and for each of those cases you will have to contend with the same number of permutations and memory sizes. It isn't easy to cut out any of the versions. The smallest markets are for 386/387 packages, but it is there that you will see important performance advantages over the older generation of real-mode compilers.

Each of the vendors had a chance to respond to this review, and I found their comments on this issue to be revealingly unrealistic. One suggested bundling the hardware and software; another suggested distributing source code. A third pointed out that their users compiled and executed code only on the same PC. I certainly believe that none of the current users of these packages are distributing executable code—how could they under current conditions? That is hardly evidence that they wouldn't like to, however!

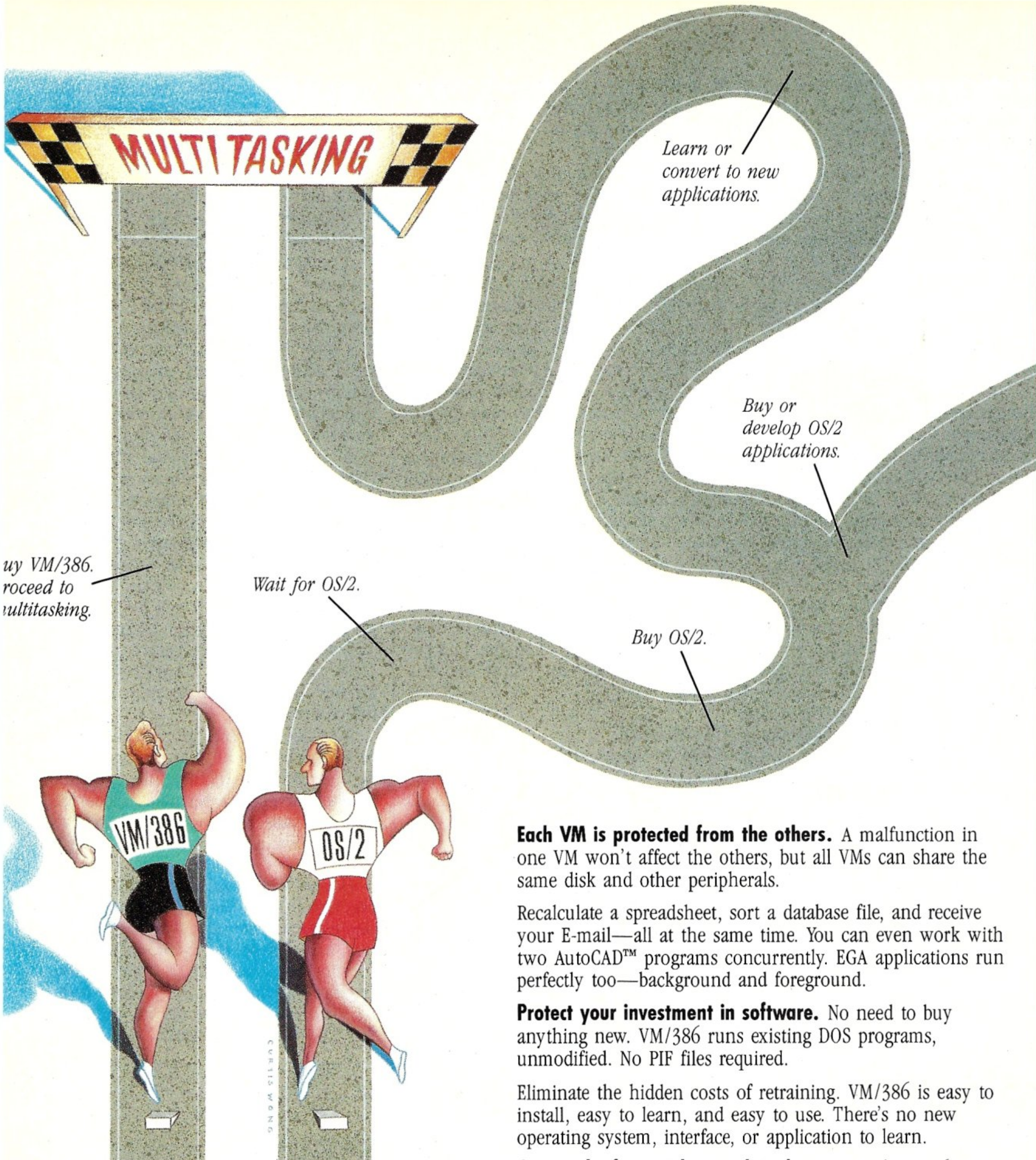
Third, the NDP compiler offers data compatibility problems, too. Without an effective means of reading nondelimited (stream) files, you can't read the binary data produced by other compilers. And that means your protected-mode program can't interchange data with the real-mode package you are probably already selling.

If you use either the F77L-EM or NDP compilers, the purchaser of the 8-MB version of your package may be loading six or seven diskettes of uninitialized common, and using up that much space on the fixed disk. Lahey informs me that the outcry from beta-test users is so great that a packing option for blank common would be provided, but it did not arrive in time for review. The NDP compiler does allow local variables to be declared as AUTOMATIC, but this introduces nonstandard syntax where it is not required. Praise to LPI and SVS for getting this right.

Debugging

For optimizing compilers on a relatively new architecture, it is perhaps unfair to expect an ideal debugging environment. However, the NDP and LPI compilers set new lows in this department. Even with optimization off and debugging flags set, the run-time executives for these two compilers may report an error—but provide no information about what line or subroutine was executing when the error occurred. What you do get is a location in the run-time library that caught the error, but no traceback to your own code. Actually, it is more likely that the computer will mysteriously reboot or hang beyond the reach of *<Ctrl-Alt-Del>*. You will have to reboot anyway because real-mode memory is not always returned to DOS. F77L-EM includes a simple source-level debugger that disposes of most errors quickly. Even without the debugger, however, the messages are clear and accompanied by a subroutine traceback. If the /l switch is invoked at compile time, then

A single benchmark reveals little about any serious performance issue, but an examination of the timings does tell a coherent story.



VM/386™

Each VM is protected from the others. A malfunction in one VM won't affect the others, but all VMs can share the same disk and other peripherals.

Recalculate a spreadsheet, sort a database file, and receive your E-mail—all at the same time. You can even work with two AutoCAD™ programs concurrently. EGA applications run perfectly too—background and foreground.

Protect your investment in software. No need to buy anything new. VM/386 runs existing DOS programs, unmodified. No PIF files required.

Eliminate the hidden costs of retraining. VM/386 is easy to install, easy to learn, and easy to use. There's no new operating system, interface, or application to learn.

Get on the fast track to multitasking. Call (408) 986-8373 for more information or to order VM/386. Everything else is just the runaround.

/M/386™. The Fast Track To Multitasking.

un smart. Run efficient. Run VM/386 on your 386-based C and start multitasking <u>now</u>! No detours, no waiting, o runaround.

rue multitasking. VM/386 uses the virtual 8086 mode of ıe 80386 processor to create many Virtual Machines (VMs) ı one computer.

ou can load a different application into each VM. Each M has up to 640K RAM, plus its own DOS, CONFIG.SYS, UTOEXEC.BAT and memory-resident programs along with s applications. Tailor each VM to your needs. You have omplete control.

System Requirements

80386-based computer such as IBM® PS/2™ Model 80 or COMPAQ® DESKPRO 386,® or 80286-based computer with Intel® Inboard™ 386/AT.

One 1.2 Mb (5 ¼") or one 3 ½" microfloppy disk drive.

One hard disk drive.

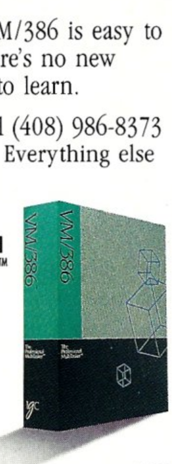
DOS 3.0 or later.

2 Mb RAM recommended.

Supports monochrome, CGA, EGA, VGA, and Hercules™ monitors.

Not copy protected.

The Professional MultiTasker™



IGC
4800 Great America Parkway
Santa Clara, CA 95054
(408) 986-8373

IGC

VM/386 is a trademark of IGC. IBM is a registered trademark of International Business Machines Corporation. COMPAQ and DESKPRO 386 are registered trademarks of Compaq Computer Corporation. Intel is a registered trademark and Inboard is a trademark of Intel Corporation. Hercules is a trademark of Hercules Computer Technology. AutoCAD is a trademark of Autodesk, Inc.

CIRCLE 54 ON READER SERVICE CARD

the line number at the point of detection is available and reported also. SVS stands at an intermediate point. There is no traceback on errors, but a mechanism for locating the line number of an error is provided. Because use of this mechanism required running the program twice under the debugger and didn't always work, I would grade it as barely adequate. Apart from that failing, the SVS source-level debugger is very useful. For myself, debugging support is the critical factor in favor of the Lahey compiler.

For the Future

As this review is written, the oldest of these products has been on the market 10 weeks, the youngest only two. There is an active effort by all developers to improve the compilers, and they have told me about the more significant features they expect to have implemented by late spring.

For Lahey these include a full 386 compiler competitive in execution speed to the other compilers reviewed here, a reduced executable module size (by packing at least blank common) and Weitek 1167 support.

LPI plans to allow mixed character and numeric variables in common, and to add a *DO WHILE* construction, Weitek support, a source-level debugger (joint project with Phar Lap) and a routine to get the DOS command line and environment.

MicroWay has a long list of plans for the NDP compiler including reduced size of executable files (better treatment of uninitialized common and greater granularity of the run-time library), generation of object rather than assembly code, improved local optimizations and utilization of the NDP stack, an interface to TSR programs, a direct screen interface, and a trace feature.

SVS plans to add double precision complex variables, and a utility library allowing full access to over 50 DOS services; it also anticipates distributing a linker with the compiler at no additional charge.

When all these enhancements are in place, differences among the compilers will be less than they are now, and the usability of each will be substantially increased. I was particularly impressed with the commitment every developer made to Weitek 1167 support, and regret my inability to test it for this review.

Conclusion

Which of these compilers would I recommend? I can't make an unconditional recommendation because each of these compilers has advantages and each has problems. Lahey is clearly the standout for ease of use, debugging, completeness, and portability of compiled programs. It is a marvelous development compiler and the only one to work with the 286 CPU. But that portability to 286/287 machines is achieved at the expense of performance on 386 systems. For a 386 compiler the choice is less clear, although it is easy to rule out LPI and SVS if you need double precision complex. Individual missing features may not be that important if you can provide them yourself with a few lines of assembly code or C. Indeed, by the time you read this, many deficiencies will have been corrected. Before making a decision I would quiz each vendor closely about recent improvements. Possibly decisive will be which product actually compiles your code correctly. Debugging with NDP or LPI will be a nightmare, and even SVS is cumbersome. You will want to start with extremely clean code, ideally a program that has already been ported through several difficult compilers.

Actually, in spite of any start-up problems I found all these compilers to be of exceptional quality. The systems deliver a level of performance closer to that of a superminicomputer than that of a personal computer. □

Daniel Feenberg, an economist at the National Bureau of Economic Research in Cambridge, Massachusetts, has his degree in Economics from Princeton University. While Dr. Feenberg writes primarily about public finance, he also moderates the Byte Bix FORTRAN conference.

Product Information

Lahey Computer
P.O. Box 6091
Incline Village, NV 00000
(702) 831-2500

Language Processors, Inc.
959 Concord St.
Framingham, MA 01701
(617) 626-0006

MicroWay
Cordage Park
Plymouth, MA 02360
(617) 746-7341

Science Applications
5150 El Camino Real
Los Altos, CA 94022
(415) 960-5931

NetCommander

A simple alternative to true local area networks

by Thomas Pasquale

Although local area networks (LANs) have become the ultimate solution to many computer problems in today's market, they are often economically infeasible for limited budgets. Recognizing the magnitude of this problem, Digital Products Inc. (DPI), operating on the belief that the importance of peripheral sharing and file transfers greatly outweighs that of file- and data-sharing, has developed an inexpensive yet reliable product called the NetCommander. DPI labels the NetCommander a sub-LAN because it lacks data- and file-sharing capabilities. DPI also notes that the NetCommander lacks the complexity of a full-fledged LAN.

DPI manufactures four models of the NetCommander: NC6, NC10, NC16, and NC32. The number in each model name refers to the number of ports with which each NetCommander is equipped. For example, the NC6 contains six serial ports or four serial and two parallel ports. The NC10 is available with 10 serial ports, or with six serial port and four parallel ports. The NC16 is pre-configured for 16 serial ports or 12 serial and four parallel ports. All models are available with any combination of serial (RJ-45 or DB-25) and parallel (DB-25) ports, and can be configured with either high-speed long-distance or RS-422 modular ports. All models feature the capability to daisy-chain other NetCommanders, as well as various buffer sizes, port selection, port contention, printer sharing, modem sharing, e-mail, file transfer, data collection, and a host of other features. The cost of the NetCommander ranges from $150 to $250 per node (depending on the size of the buffer and the type of port), compared with a typical cost of $600 to $800 per node on a true LAN.

As an employee of Academic Computing at La Salle University, I recently had the opportunity to install and work with a NetCommander model NC16. It is a compact unit, measuring 17 by 10 by 4-inches, and its front panel contains an on/off, restart, and reset switches. There are 16 RJ-45 serial ports at the rear. A setup program is supplied, along with an owner's manual, an applications guide, and a tutorial on networking and printer sharing.

Hardware Setup

The Owner's Manual adequately describes how to install and configure the NC16. Creating the actual link between the NC16 and your hardware requires some knowledge of the different devices involved. For example, a modem demands a cable for communication to the NetCommander that is different from that of a PC. A simple setup of 14 PC workstations and two printers should take no more than 30 minutes to an hour to attach to the NC16 box. (Actually, DPI's Print-Director better suits this setup.) However, at La Salle University, our setup of five PCs, four terminals, three printers, two modems, one print spooler, and one plotter took a little more time to install because of the number of different devices involved. DPI stocks all the cables you will need. Realize that the cabling for the NC16 with RS-232 DB-25 serial ports costs $300 more than the NC16 with RJ-45 ports. The only difference between the two is that DCE/DTE selection is done inside the RJ-45 to RS-232 converter; it cannot be done in the software configuration that is discussed later.

Keep in mind that, if you wish to place a parallel printer on the sub-LAN, you are going to need a parallel-to-serial output converter. A converter costs about $100.

Software Configuration

After all the devices are connected to the NC16, the software configuration may be attempted. Although it is a tedious task, once the NetCommander is configured it rarely needs modification. The purpose of the configuration process is to set the parameters—such as port name, baud rate, parity, and handshaking—of each port. All parameters for each port on the NC16, except for the baud rate, are programmable on an individual basis. Baud rates are set by the NC16 in the following six groups:

Group	Port
1	0
2	1
3	2,3
4	4,5,6,7
5	8,9,A,B
6	C,D,E,F

Because the baud rate within each group had to be the same, the devices at La Salle were arranged in the following fashion: five PCs in groups 1 and 4, four terminals in group 5, three printers and one plotter in group 6, two modems in group 3, and one print spooler in group 2. With such a configuration, the NC16 must understandably do a great deal of protocol conversion so that a port using one set of

parameters is able to communicate with a port using a completely different set of parameters.

The NC16 software configuration may be accomplished in two different manners: either through the AutoInstall program or through the Menu Mode program in the NetCommander's ROM, accessed by a terminal emulator. The AutoInstall program (which requires an IBM or IBM-compatible PC, XT, or AT connected to port 0 of the NC16 and running MS-DOS or PC-DOS) is by far the easier of the two methods because it allows a single device to be completely configured before advancing to the next one.

The Menu Mode program, on the other hand, is designed to set a single aspect of all devices at the same time so that first the baud rates of all devices are set, then parity is set, then handshaking, and so on. Using the AutoInstall program is more beneficial if, perhaps later, a printer is replaced by a modem on the sub-LAN. In the AutoInstall program, the user chooses the printer, changes it to a modem, and adjusts the corresponding settings for a modem. Using the Menu Mode program menus, the user may have to struggle through 10 different options to perform the same task.

A final reason for using the AutoInstall program is that it saves the configuration on disk before loading it into the NC16's memory, whereas the configuration set by the Menu Mode program resides only in the NC16's memory, waiting for something to go wrong. Infrequently, a section of memory does get zapped. Running the AutoInstall configuration program will take about five minutes to remedy this situation, but reconfiguring it through the Menu Mode program is another story.

Incidentally, the AutoInstall program does not have to be run again if power to the NetCommander is lost. A section of memory containing the permanent configuration is battery powered and should remain intact without electrical power for over a month. If the configuration is damaged, however, AutoInstall must be rerun.

Operation

Action by the NetCommander is keyed to five programmable command/control characters: end-of-job, cancel, break, command, and end-of-page character. Rarely used characters, such as ^\, ^P, ^B, ^T, and ~ are good suggestions for these characters since they will be intercepted by the NetCommander and they will have no other function.

The command character is the most important of the command/control characters because, when followed by the name of a port, it is used to queue a job to a selected port. For example, if the command character is ^T and you wish to use a dot matrix printer previously named DOT, a ^TDOT is echoed through the communications port of your machine to select it. On a PC, the use of batch files makes the selection of ports user friendly. In the example given above that selects the dot matrix printer, a batch file, DOT.BAT, could be created containing the single line

```
ECHO ^TDOT > COM1
```

This command would work even if the port name was DOTMATRIXPRINTER, provided no other port name begins with DOT.

If the same name is given to several ports, the first available port is chosen. Note that because all data is sent to the NC16 through the PC's serial port, output to be printed must be redirected to the serial port after a parallel printer is selected. This is accomplished (typically in the AUTOEXEC.BAT file) with the line

```
MODE LPT1:=COM1:
```

The normal means to terminate a job is through the end-of-job character. Since this character is not always sent to the NetCommander because a PC might be turned off in the middle of a job, the NC16 also provides a user-determined timeout to avoid cases of ports being tied up indefinitely. At first, the length of the timeout chosen is a guess at the longest predicted period of inactivity between two devices. As time goes on, the length of this period will become obvious. For printers, the timeouts tend to be short; for terminals, they tend to be longer.

In addition to the Command/Control characters, other features that NetCommander provides are strings printed at the start of a job, at the end of a job, when a port is busy, and when a port is ready. These strings can be printed either on the sending or receiving device. It also supports a timeout for normal mode, a timeout for graphics mode, and a default port to attach if the NC16 is reset. All of these may be set with the advanced options in AutoInstall. Before the latest version of AutoInstall (Version 1.04), however, they were set with the Menu Mode program and would be lost if a new configuration was run.

Job Scheduling

Job scheduling is another advanced feature the NetCommander offers. When a user selects a device that is currently in use, the request is placed in a queue and is granted after the device becomes available. If a user's PC is in terminal mode when making this request, the user receives a busy message (if one is set) and another message when the device is ready. Of course, the user may choose to cancel the request by issuing the special cancel character. The Menu Mode program under the System Functions option provides a list of the job table. The table specifies the job priority (first come, first served), the status, the source port, the destination port, and the name of the source port.

Graphics Mode

In order for the NetCommander to transmit binary files or graphic characters between two devices, it must enter graphics mode, which is done by selecting a port and entering two command characters and a G. For example, a PC that wants to print a Lotus 1-2-3 graph on the dot matrix printer would first create and execute a batch file, DOTG.BAT, containing the lines

```
ECHO ^TDOT > COM1
ECHO ^T^TG > COM1
```

These commands prohibit the NetCommander from intercepting any of the 256 ASCII codes and, of course, from further port selection (the command character will now be considered data) unless a graphics-release string is sent or the graphics timeout occurs. The default value of the graphics release string is

```
++++++ ... ++++++!
```

and may be changed exclusively in the Menu Mode program. This string consists of three parameters: the graphics-release character (+), a count to repeat this character (40), and a terminator character (!).

In most cases, the graphics timeout is set to be a long period of time, so the only way left to leave the graphics mode is by issuing the graphics-release string. In consideration of constraints on time and patience (how many of us would invest the time to count out 40 or more signs and follow it with an imperative?) this string should be placed in a batch file. Otherwise, the device sent into graphics mode will probably remain there when the user leaves. Using the dot-matrix printer as an example, the file DOT.BAT is modified to read

```
ECHO +++ ... +++! >
COM1ECHO
^TDOT > COM1
```

Because the mechanism does not have the ability to poll a device to determine whether it is in graphics or

normal mode, this method has one bad side effect. If the device is not in graphics mode, the graphics release string will be printed on the output device, which is normally a printer.

File Transfers

The ability to transfer files across PCs is important in some environments. DPI sells a network software package called EasyLan, available for $100 per PC. EasyLan possesses DOS-like commands, such as EZCOPY, EZDIR, EZTYPE, and EZDEL. Each of these commands carries a remote-device name in one of its arguments. EasyLan is useful but, at $100 per PC, it is expensive.

The alternative to EasyLan is simple: use your own communications software. For example, occasionally I need to copy files from a 5¼-inch floppy disk to a 3½-inch disk. The machine with the 3½-inch disk drives, a portable IBM-compatible Zenith Z-181, temporarily replaces a PC on the sub-LAN. The Mirror communications program is run on this machine and on one other PC attached to the NetCommander. After the connection between the two computers is created, it is necessary to send the computers into graphics mode because a file that is going to be transferred may contain one of the five special characters intercepted by the NetCommander. Popular products, such as Crosstalk and Mirror, support software protocols, such as kermit and xmodem. The Z-181 runs a command similar to *rk b:* that tells Mirror to use the kermit protocol to get ready to receive files and where to place them. The other PC runs a command such as *xk b:*.** that tells Mirror to use the kermit protocol to transmit all the files on the B drive.

Modem Handling

Modems cause several problems for the NetCommander. Consider the earlier configuration that placed two modems in ports 2 and 3. Although the baud rates may be changed quickly through the Menu Mode program, there are times when both modems are needed at different baud rates. No easy solution to this difficulty exists.

Another problem (or rather inconvenience) is dialing out. In a communications package such as Crosstalk or Mirror, a telephone number is placed in the program by typing NU 1231111 and dialed by typing GO. In the NetCommander setup, the settings for the communications package are for the PC and not the modem. The settings for the modem were issued back in the AutoInstall program or in the Menu Mode program. To use a modem with the NetCommander, a communications program is run and the PC is placed in local mode. Everything that is typed thereafter is sent through the PC's serial port and thus into the NetCommander.

After the port attached to the modem is selected, the task of dialing out must be tackled. Placing the telephone number in the communications program is ineffective because the terminal must be in local mode. The attention code that the modem uses to dial out must be physically typed. With a Hayes-compatible modem, ATD 1231111 would be typed. Of course, most communication packages possess a script-file facility to make this process transparent to the user.

E-Mail

Finally, the NetCommander has a simple electronic mail feature. Typically, two users will be in terminal mode when using e-mail. To send mail, a user selects a port and types a message header, <Ctrl-B>START, the message, and <Ctrl-D>.

Summing Up

Overall, the NetCommander NC16 warrants a very good to excellent rating as a sub-LAN because it effectively ties together a number of different devices in a harmonious way. In addition, the only programming needed is the creation of a few small batch files, which makes using the NetCommander very convenient. You would be hard pressed to beat what NetCommander accomplishes at its price. □

Thomas Pasquale is an academic consultant and programmer for the Academic Computing Center of La Salle University in Philadelphia. He has a degree in Computer Science and Physics from La Salle.

Did you find this article particularly useful? Circle number 9 on the reader service card.

Product Information

Prices vary depending on the connectors and other options selected. Please contact Digital Products for details about configurations.

NetCommander	
NC6	$1,295–$1,795
NC10	$1,695–$3,195
NC16	$2,695–$4,295
NC32	$4,950–$8,225

Digital Products Inc.
108 Water St.
Watertown, MA 02172
(617) 924-1680

by Patrick H. Corrigan

Database Service, File Service, and Server-Based Applications

Making assumptions can be dangerous, and this is especially true when working with LAN systems. Unfortunately, many LAN buyers, users, and integrators often make decisions based on erroneous assumptions, incomplete data, press releases, and false information. When a major company makes an announcement concerning a new, not-yet-in-existence product or capability, that announcement often becomes the current "Hot New Topic" and the erroneous assumptions begin.

For example, when IBM executives announced their Token-ring network (several years before it was shipped), they indicated that it would run on twisted-pair wire. Many writers and industry observers assumed that this meant cheap telephone wire. The early trade press articles almost invariably referred to "cheap twisted pair versus expensive coax cable." Unfortunately, the reality is quite different. Technically, the Token-ring will run on certain, specified phone wire, but only for limited distances. Relatively expensive, data-grade, shielded, twisted-pair cable is usually required for Token-ring networks. Assuming that "twisted-pair" meant telephone wire lead many people to underestimate the cost of implementing Token-ring LANs.

We are currently facing a similar scenario. Two of today's "Hot New Topics" are Database Service and Server-Based Applications. Again, as with previous "Hot New Topics," the observers are beginning to make assumptions. The main assumptions are that database service is inherently better and faster for shared database access than file service, and that server-based database servers are better than other approaches. Before we discuss the relative merits of these two approaches, let's discuss what they are and what they do.

File Service

Currently, the standard approach for retrieving data from a network server is file service. Network file servers control access to shared files, among other tasks. These files can be program files, text files, database files, or any type of file. With file service, a workstation PC sends a request to the file server, which then downloads the requested files, or portions of them, to the workstation PC's RAM. With many programs and files, the whole file is loaded into RAM. When certain files are too large, however, only portions of the file are downloaded. With most database files, for example, only the file blocks that contain the requested data records and associated indexes, if any, are downloaded. In most cases, access to that portion of the database is then "locked" to other users. The workstation PC can modify and/or update the specific locked records, then upload the file blocks containing them back to the server, at which time they will be unlocked to other users. With file service, database processing takes place at the workstation.

Database Service

Database server systems use the workstation PCs for the database "front end," or user interface, and another machine on a network, often the network file server, for the "back end" database processing. This means that the processing of database requests and updates is performed on the database server rather than at the user's workstation. With this approach, only individual records are transmitted from the database server to a workstation. Because individual records are being sent to and from the workstation, traffic on the network may be lessened.

It sounds as though database service would be faster and more efficient for database processing than file service, right? Well, maybe yes and maybe no.

A database server can be implemented in several ways:

1. It can be a separate, secondary process on a network file server;
2. It can be a Value-Added Process (VAP) on a file server, working as part of the LAN operating system;
3. It can employ a coprocessor in a LAN file server;
4. It can use a dedicated PC on the LAN;
5. It can be a separate background process on a designated LAN workstation; or
6. It can employ a coprocessor in a LAN workstation.

All of these approaches can be considered server-based applications, and each has its own advantages and disadvantages. Remember, "server-based" does not necessarily mean "file server based," although that is the current prevalent usage of the term.

1. Database Service as a Second Task on a File Server. This is the approach that is currently receiving the most attention by the computer press. The server CPU is shared between the file server function and the database server function. Using OS/2 and the Microsoft LAN Manager, for example, database service would be a secondary, separate process running on a network file server. The basic idea is that the file server is the "logical" place for database service to take place. What is not generally discussed, however, is how much impact this approach will have on overall performance. With a slow network, a fast server, and little traffic, this approach may be fine. But what about a fast, busy network? Will sharing the server CPU between file service and database service create a performance bottleneck? The answer is probably, "Yes."

2. Database Service as a Value-Added Server Process. A Value Added Process (VAP) is an extension to the file server software. (This is one approach Novell has taken). With a VAP the server performs additional tasks that are integrated with the file server software. Although tighter integration of tasks could result in less overhead, this approach seems to have the same potential performance problems described above.

3. Application Coprocessor in the File Server. With this approach, a separate processor board is installed in the file server expansion bus. Database requests would be routed to the coprocessor, thereby easing the load on the main CPU. All other things being equal, using an application coprocessor in a file server would probably provide the best performance of the database servers. It would not use file server CPU time for database processing, and it would not add to traffic on the LAN cable.

4. Dedicated Database Server. With a dedicated database server, i.e., a separate PC used only for database processing, traffic between the database server and the workstations can be reduced. Database files could be stored on a hard disk on the database server PC or on the file server. However, if the files are stored on the file server, which is usually desirable for both security and backup reasons, the database server has to update the files on the file server, creating more LAN traffic.

5. Background Database Server on a Workstation. Under OS/2, a workstation PC could be used as a database server, with the server functions being a background task. This would have most of the attributes of a dedicated database server, but would provide lower performance due to the sharing of the workstation CPU, bus, and network interface.

6. Application Coprocessor in a Workstation. A coprocessor in a workstation PC would probably have about the same overall effect on performance as the separate database server. Because the expansion bus and network interface are being shared, performance of the host workstation could be affected in a heavy-traffic situation.

Is Database Service Better?

One additional performance factor to consider is the database server itself. Multiple requests and updates must be queued by the database server, while update processing with file service is provided by the workstation PC. With a busy system, the database server itself can become a bottleneck. Without a database server, however, database processing is performed by multiple CPUs, effectively distributing the workload.

A potential problem with distributed database processing is that any station could corrupt the database. Since, in the case of database service, all database processing is being handled by a single CPU, there is less chance of file corruption.

System integrity is another factor to consider, especially with applications and/or coprocessors running in the file server. What if a server-based application "hangs up" the server? Applications have been known to hang-up single PCs, and they could also hang-up a file server running a server-based application.

While many database server systems have been announced, only a few are actually available for PC LANs, and very few benchmark tests have been published. The few benchmarks I have seen gave mixed results. In any case, be wary of making assumptions. It's an easy way to get burned.

Now, on to other LAN-oriented topics.

Do Simple LANs Require Less Management?

After seeing my outline for a Novell System Manager course, one company decided to purchase a "simple" LAN for its three-PC office. This company assumed that the simple, inexpensive LAN would not require any management. After many attempts by both the company and the LAN dealer to make the system work properly, company management was ready to throw the LAN out. In fact, because of this bad experience, company management is nearly convinced that LANs don't work.

The truth is all LANs require proper installation and proper management. Simple, inexpensive "resource sharing" LAN operating systems are not only less expensive than systems like Novell NetWare or Banyan Vines, but they are also less sophisticated and usually far less capable. This means that more management, not less, is often required.

LAN Connectivity

Connectivity and interoperability of dissimilar systems across LANs is another "Hot New Topic." Novell, Inc., recently announced a version of its NetWare LAN operating system that uses a DEC VAX running the VMS operating system as a LAN file server. This product is designed to provide PC LANs transparent access to VAX files (i.e., VMS files will look like DOS files to

DOS, and DOS files will be stored as VMS files on the VAX). Although VMS NetWare is only supported on Ethernet, NetWare bridges and terminal emulation software will allow users on other networks, such as IBM Token-ring or ARCnet, to access the VAX.

This is a first step in Novell's plan to make NetWare a universal transparent link between dissimilar systems. Expect similar announcements concerning the Apple Macintosh, since Novell has licensed the Appletalk File Protocols (AFP) from Apple. Also, at a recent Novell connectivity seminar, Phaser Systems of San Francisco unveiled software that provides NetWare LANs virtual file server capability on IBM mainframes.

Although VMS NetWare is only supported on Ethernet, bridges and terminal emulation software provides access to the VAX.

TOPS, a Sun Microsystems Company, also recently announced software to allow DEC VAXes to act as network servers on TOPS networks using Ethernet. TOPS currently provides software that allows Macintoshes, PCs, and Sun Microsystems workstations to be networked together. □

Patrick H. Corrigan is a partner in The Corrigan Group—Information Services, an independent consulting firm specializing in local area networks and office automation based in Corte Madera, California.

Did you find this article particularly useful? Circle number 7 on the reader service card.

Product Information

Novell, Inc.
122 East 1700 South
P.O. Box 5900
Provo, UT 84601
(800) 379-5900

by A. G. W. Cameron

Recent News About TEX

Created for material rich in mathmatical expressions, TEX is a text formatting program designed to prepare technically complicated material for a typesetter. "Proof" copies of the output can be obtained from laser and dot-matrix printers (this frequently becomes the final output as well).

TEX was developed by Donald Knuth of the Stanford University Computer Science Department. His primary goal was to make his books on computer algorithms look "beautiful." A regular industry has grown up around the use of TEX; the program started on mainframes and has been ported to mini- and microcomputers. The primary goal of all this has been to make TEX a standard program, so that the ASCII output of the TEX program running on a source file will produce an output file that can be further processed by any computer for any output device (the output file is called a device-independent or DVI file).

The microcomputer community has benefited from healthy TEX competition between two vendors, Addison-Wesley and Personal TEX. Each of these companies sells versions of TEX for IBM PC/XT/AT clones and for the Macintosh.

PostScript Drivers

One of the main objectives in producing wide standardization of TEX is that the output of a given source file should look the same, apart from printer-dependent limitations, no matter what computer implementation of TEX is used. This requires a standard font set, and so Donald Knuth authored a program, METAFONT, that acts upon the values of a large number of parameters to transform standard input files into the characters of a distinctive font. Thus, for example, a relatively small number of parameter changes would suffice to transform a roman style into italic, or boldface, or a slant version that is recognizably part of the same font family.

The striking thing about METAFONT is that slight changes in other parameters can change the basic flavor of the font family altogether, even for the same input files. Thus, a serif font can be converted into a sans-serif font, or the height of the central body of the characters (the "x" height) can be varied relative to the ascenders, and so on. Some people apparently feel that this procedure represents a breakthrough in typography, whereas many font designers believe that the resulting fonts have no soul.

During the development of METAFONT a preliminary set of "almost computer modern" fonts was produced and widely distributed to the TEX community, and more recently the final set of "computer modern" fonts was issued. These "am" fonts are now gradually being displaced by the new "cm" fonts, accompanied by much confusion. The two fonts differ in only minor effects, and many people regard them as basically utilitarian and not especially pleasing to the eye. Now the basic standardization intended for the computer modern fonts is being eroded on several fronts.

The first variation was the adaptation of PostScript laser printers, such as the Apple LaserWriter for TEX output. These PostScript printers have been provided with from a few to many internal fonts by Adobe Systems, the creator of PostScript. There are now two PostScript drivers available for use with TEX: the ArborText DVILASER/PS program ($225) sold by both Personal TEX and Addison-Wesley, and the Personal TEX PTI Laser/PS program ($195). Each takes a DVI output file and converts it to a PostScript file. An accompanying small program can be used to send it to the PostScript printer. In my system, I simply take the PostScript file and send it over Ethernet to my Sun workstation, which queues it to the LaserWriter.

Each PostScript driver can use the internal PostScript fonts, which provide the TEX Font Metric (TFM) files (lists of character widths, heights, and kerning and ligature information) that enable TEX itself to typeset material using these fonts. The original LaserWriter had Times Roman, Helvetica, and Courier font families, each with the ordinary, oblique, boldface, and boldface oblique versions. Later PostScript printers typically have about 35 font families. The use of these fonts with TEX makes for much more varied and interesting typography, but if you want to use these fonts you will probably lose the ability to send your TEX source file to a friend or collaborator in a form that can be printed out.

There are significant differences between the two PostScript drivers. The ArborText version requires more fussing to set up; you must tell it (via information files) where the TFM files are, where the pixel files are (the files containing the dot patterns for the characters), and what sizes of pixel files are available for each font. Unless you introduce new fonts to your system, however, this need only be done once. Both drivers have an extensive list of options that can be set interactively. Both drivers allow the merging of PostScript graphics with the TEX text, but the ArborText manual gives a much more extensive set of instructions.

The ArborText driver is marginally the faster of the two and seems to do a better job of managing the fonts downloaded into the LaserWriter. However, I have had the ArborText driver hang up on me on a number of occasions, so it appears not to be free of bugs even though it is about two years old. The Personal TEX driver has to be told how to manage the fonts, either to keep them all in the LaserWriter's limited memory (which causes the driver to quit if there are too many fonts) or to reload the fonts after outputting each page (which can make enormously long PostScript files, taking up a lot of disk space and requiring an especially long time to download to the LaserWriter at 9600 baud over a serial line).

Bitstream Fonts

The newest TEX fonts come from Personal TEX, which has adapted the Bitstream fonts. Bitstream fonts are being widely used in place of Adobe fonts in the cloning of PostScript, and there are a lot of pleasant ones from which to choose. Font names can be copyrighted, but not font shapes, so the practice in the industry has been the

wholesale copying of font types and shapes under alternative names. Thus the Times Roman of Adobe is the Dutch of Bitstream and the Helvetica of Adobe is the Swiss of Bitstream. Adobe licenses its fonts from several sources, as does Bitstream, and thus the fonts have the same name when the source is the same. Bitstream also designs many of its own fonts.

Both Adobe and Bitstream use font outlines. In the generation of a bit-mapped character, the outline is scaled to the desired size (and in the case of Adobe it also can be rotated), and then the outline is filled in with dots. Both companies use sophisticated algorithms to try to get good-looking bit pattern fits to the outline shapes. Adobe does this every time one of its internal characters is used, so that the dots are generated directly into their positions on the output page. The Bitstream outlines will presumably be used in a similar way in the forthcoming PostScript clones. But in the Personal TEX Bitstream fonts, the bit-mapped characters are generated as separate pixel files for every desired size by the COTOPX (Compressed Outline to Pixel) program. This program first generates "property list" files that are converted to TFM files; then it generates a particular form of pixel file called PXL, and this in turn is converted into a packed form called a PK file. Any standard TEX output driver for any device can use the TFM and PK files just as it would these files for a standard TEX font, and thus these Bitstream fonts can be used on standard laser printers and dot matrix printers as well as in PostScript printers. But you cannot rotate the pixel patterns on a PostScript printer without treating them as explicit graphics images.

Both Adobe and Bitstream font outlines can have their shapes manipulated in various ways. The characters can be slanted through a wide range of angles, distorted by horizontal or vertical stretching, and the inter-character spacing can be varied (on the Bitstream fonts).

The Bitstream compressed outline files can also be used to generate two additional "complementary" fonts. The first of these contains a lot of specially accented characters such as those used in European languages, together with a varied collection of additional accents and sets of smaller numerals in raised and lowered positions that can be used to print fractions formed with the slanted slash mark. The second complementary font contains graphics symbols that have a large overlap with the familiar set of graphics symbols available on PCs and clones.

The basic Fontware package from Personal TEX contains the COTOPX and related software for the generation of pixel fonts, together with the Swiss family of four fonts for $195. There are 19 additional Bitstream font families that can be purchased for $195 each.

Figure 1 shows examples of four Bitstream font families, together with

The output should look the same, no matter what implementation of TEX is used.

Computer Modern Roman for comparison, all at 12 points so that the details of the styles can be seen. These Bitstream fonts are representative of the variety of ordinary fonts available, although the complete set of fonts contains some much more extreme examples.

The manual that accompanies the Bitstream font package from Personal TEX has been written by Mike Spivak, and it is especially clear and interesting.

Figure 1. Four Bitstream font families

This is an example of the Computer Modern roman font. *It has an italic version,* **and a boldface version,** *and it lacks a boldface italic version but has this separate slanted version instead.*

Bitstream calls this the Dutch font, but it is normally known as Times Roman. Notice that it has serifs. *It has an italic version,* **also a bold version,** ***and a bold italic version.***

Bitstream calls this the Swiss font, but it is normally known as Helvetica. Notice that it does not have serifs (it is a sans-serif font). *It has an italic version,* **it has a bold version,** ***and it has a bold italic version.***

This is Bitstream Charter, which Bitstream designed itself. It has nicely rounded characters with serifs, and it has become my favorite font family. *It has an italic version,* **it has a particularly nice bold version,** ***and it has the expected bold italic version.***

This is Futura Book. Notice that it is a sans-serif font with a smaller ratio of "x height" to the height of capital letters than is true in the Swiss font. *It has an italic version,* **it calls its bold version its heavy version,** ***and so naturally the fourth version is called heavy italic.***

Tables To Die For

TEX has the versatility and flexibility to enable you to put anything you want, anywhere you want on the page. But actually doing it is something else again. Many TEX constructions are very involved and are unlikely to be attempted by anyone but an expert. Tables are particularly difficult, since the TEX rules for alignment are hard to master. I have had a lot of questions from secretaries who have become quite frustrated trying to get a table to come out properly.

This was Mike Spivak's motivation in developing a huge macro package which he calls Tables To Die For (T2D4), which is sold by Personal TEX ($125). This macro package is so large

that it is difficult to use it in conjunction with other very large macro packages, and probably impossible to use with the particularly large macro package LaTEX. The great size is a result of the fact that TEX is a very wordy computer language, so you must write out a lot of stuff in order to accomplish anything complicated such as a table.

There are a tremendous number of details that can be varied in the construction of tables, and a few major necessities like getting tables to break over pages. Therefore, even though the process of setting up a given table using T2D4 is much simpler than trying to imitate the procedures used by Donald Knuth in *The TEXbook*, the bible of the TEX industry, the fact that there are a very large number of different T2D4 procedures tends to be confusing to anyone but an expert.

> *TEX has the versatility and flexibility to enable you to put anything you want anywhere you want.*

T2D4 is very useful to have around for those occasions when you want to make tables. Its very large size as a macro package may mean that you will prefer to use it in a supplementary fashion, to prepare your tables apart from the rest of your text.

MacroTEX

MacroTEX ($200) is a macro package prepared by Amy Hendrickson of TEXnology, Inc. It is intended for general use and thus there are a number of different modules intended for different purposes. The package has been set up so that each module is read into TEX's memory only when it is needed, so if you only want to use a small number of the modules, you will suffer relatively little interference with the use of other large macro packages at the same time.

One of the modules is concerned with table preparation, and thus the use of MacroTEX is, to some extent, in competition with T2D4. In my opinion

the table-making function is easier to use with MacroTeX than with T2D4, but there is less flexibility in fiddling with the fine details. Major necessities, such as breaking tables across pages, are handled properly by MacroTeX.

A very large macro package used by many people is LaTeX, which is also a general purpose package. I have never used it, because many of the familiar TeX definitions used in the plain macro package are eliminated, and the resulting program can no longer perform a lot of operations that may be desirable (even though the package may be easier to use). What is striking about MacroTeX is that the definitions of plain TeX are almost entirely untouched, so you do not lose flexibility.

Like LaTeX, MacroTeX is concerned with the preparation of manuscripts in different styles. The styles supplied are documentation, book, report, letter, and note. If you want to edit these macros to change details of the styles, this should be much easier to do here than in LaTeX. MacroTeX also can be used to prepare a table of contents, lists of tables and figures, cross-references, indexes, glossaries, bibliographies, and end notes. Automatic equation numbering and cross-referencing is another feature. Variations of page parameters are easier to do than with plain TeX.

Amy Hendrickson has prepared an excellent manual with lots of illustrations. There is a great deal to be pleased with in this package. □

A. G. W. Cameron is Professor of Astronomy at the Harvard-Smithsonian Center for Astrophysics.

Did you find this article particularly useful? Circle number 8 on the reader service card.

Product Information

Addison-Wesley Publishing Co.
1 Jacob Way
Reading, MA 01867
(617) 944-3700

Bitstream Inc.
Athenaeum House,
215 First St.
Cambridge, MA 02142
(617) 497-6222

Personal TeX Inc.
12 Madrone Ave.
Mill Valley, CA 94941
(415) 388-8853

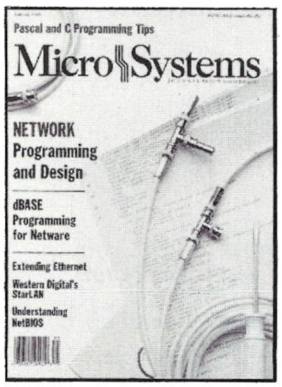
TeXnology, Inc.
57 Longwood Ave.
Brookline, MA 02146
(617) 738-8029

Tell Us What You Want - And Get What You Need

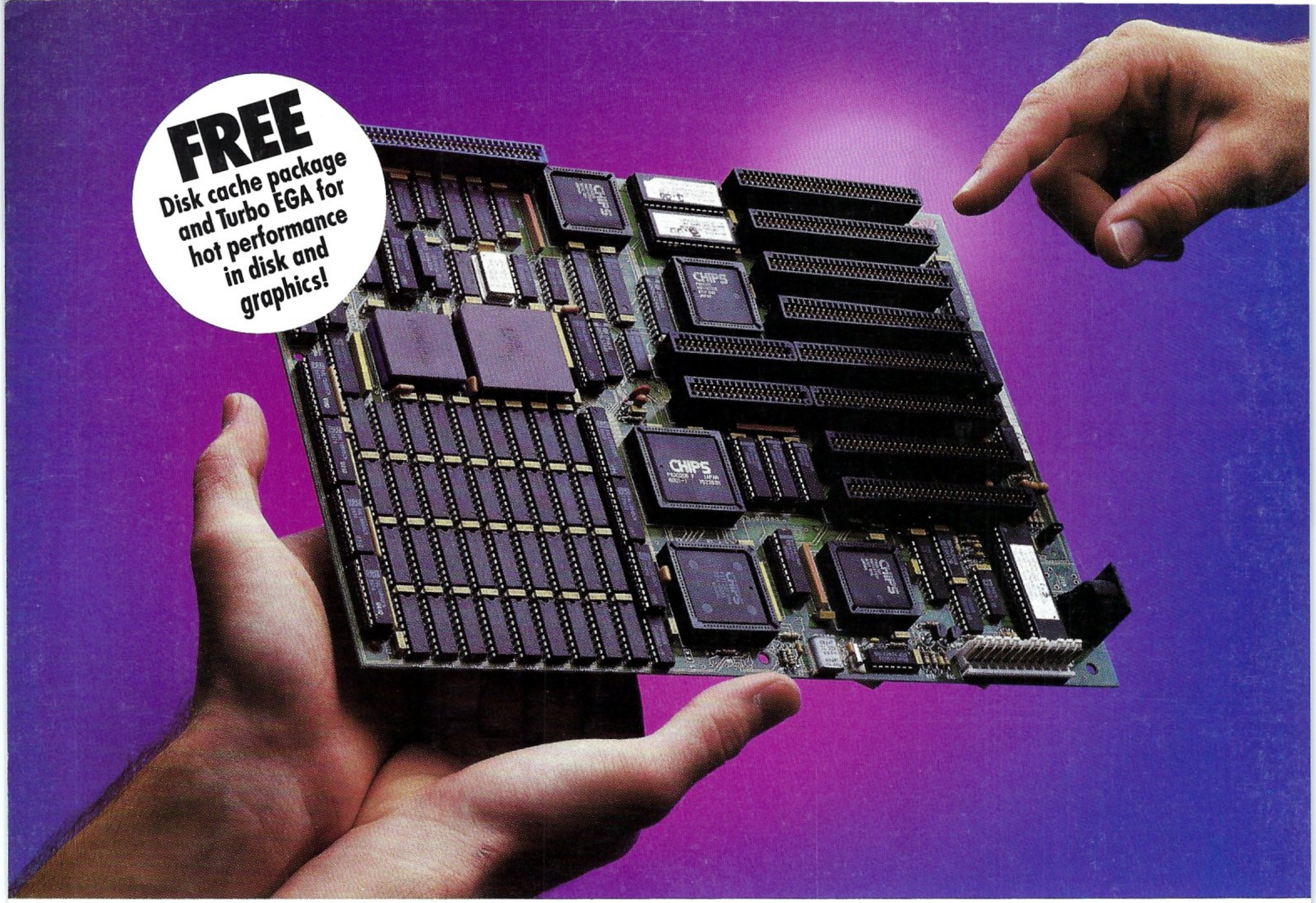
Now you can get the product and service information you need with reader service, thanks to *Micro/Systems Journal.*

By filling out the reader service card, you receive the information you need from our advertisers. This includes facts on specific products or additional literature on their services. *And* when you answer the short editorial questions, you tell us what you want to read about in future issues. Finally, by responding to our advertisers through reader service, you contribute to the growth and success of *Micro/Systems.*

To get your free information, look for the reader service numbers listed at the end of articles and ads. Then simply fill in your name and address on the card opposite page 40, answer the editorial questions, circle the number(s) of your choice, and drop the card in the mail. It's that simple. And it's absolutely free.



Advertiser Index



CONVERT YOUR PC, XT OR AT INTO A HIGHER FORM OF LIFE!

386 MOTHERBOARDS FOR 386 SPEED

Don't let your PC give up the ghost—Hauppauge has just arrived with a new spark of life: the 386 MotherBoard.™ Far more advanced than an accelerator card, our line of MotherBoards grace your PC, PC/XT, PC/AT or compatible with speeds equal to the IBM PS2 Model 80. And faster. Because we've built in 1 Megabyte of high speed RAM and a 387 math coprocessor socket for speeds that make users humble with awe.

OS/2 Compatible. To ensure a long, fruitful life, our 386 MotherBoard is compatible with the PC/AT (BIOS and I/O)—so you can run the new generation of DOS, OS/2. Our Board also runs Windows/386, UNIX V and PC-MOS/386. For more power, you'll find 16-bit expansion slots that accommodate the latst I/O expansion cards. No 386 accelerator card gives you so much versatility. **Only our 386 MotherBoard gives your PC a future with unlimited possibilities!**

The Critics Applaude! *PC Magazine* awarded our Board "The Editor's Choice" for 386 Replacement Boards. *PC World* called it "the Upgrade Product of the Year."

Technical Features ▪ 16 MHZ 80386 ▪ 1 Megabyte of 100 nsec 4-way interleaved RAM ▪ PC/AT compatible I/O and BIOS for support of OS/2 ▪ Six 8-bit expansion slots ▪ two 16-bit expansion slots (four on 386 MotherBoard/AT) ▪ One 32-bit expansion slot for up to 12 Megabytes of high speed memory ▪ Battery-powered clock calendar

386 MotherBoard/PC or MotherBoard/XT	$1495
386 MotherBoard/AT	$1595
32-bit RAM Board (2 Mbytes installed; up to 4 Mbytes)	$795
16 MHz 80387 math coprocessor	$695
16-bit combination hard disk/ floppy disk controller	$245

For more information on our easy-to-install MotherBoard, call: **1 (800) 443-6284**. In New York, call **(516) 434-1600**.

Hauppauge Computer Works, Inc.
175 Commerce Drive,
Hauppauge, New York 11788

Hauppauge!

CIRCLE 105 ON READER SERVICE CARD